Peter Großöhme

Analyse von Hochverfügbarkeitsmechanismen auf der Basis virtueller Maschinen

DIPLOMARBEIT

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät
Mathematik/Naturwissenschaften/Informatik

Mittweida, Januar 2010

Peter Großöhme

Analyse von Hochverfügbarkeitsmechanismen auf der Basis virtueller Maschinen

eingereicht als

DIPLOMARBEIT

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät
Mathematik/Naturwissenschaften/Informatik

Mittweida, Januar 2010

Erstprüfer: Prof. Dr.-Ing. habil. Joachim Geiler
Zweitprüfer: Dipl.-Inform. Heino Gutschmidt

Vorgelegte Arbeit wurde verteidigt am: 1. März 2010

Bibliographische Beschreibung:

Peter Großöhme:
Analyse von Hochverfügbarkeitsmechanismen
auf der Basis virtueller Maschinen 2010. - 90 S. Mittweida,
Hochschule Mittweida - University of Applied Sciences,
Fakultät Mathematik/Naturwissenschaften/Informatik, Diplomarbeit, 2010

Referat:

Die Hardware-Virtualisierung bietet neue Ansätze für die Realisierung von Hochverfügbarkeitskonfigurationen. So lassen sich auf der Basis von virtuellen Maschinen (VM) vorhandene Ressourcen wesentlich zielgerichteter und flexibler verteilen. Kombiniert man diese mit ausfallverringernden Mechanismen auf Applikationsebene (z. B. Applikationscluster), so können damit hochverfügbare Dienste zu erheblich reduzierten Kosten realisiert werden. Ziel der Diplomarbeit ist es, auf Grundlage unterschiedlicher realer Einsatzszenarien sinnvolle Topologien zu entwickeln, die einerseits eine hohe Verfügbarkeit gewährleisten, aber andererseits auch ökonomischen Randbedingungen genügen.

Danksagung

Die vorliegende Diplomarbeit bildet den Abschluss meines Studiums der Technischen Informatik an der Hochschule Mittweida.
An dieser Stelle möchte ich mich bei Herrn Prof. Dr.-Ing. habil. Joachim Geiler für die hervorragende Betreuung sowie das entgegengebrachte Vertrauen während der gesamten Bearbeitungszeit bedanken. Ferner danke ich der managedhosting.de GmbH, insbesondere Herrn Dipl.-Inform. Heino Gutschmidt, für die technische Unterstützung sowie für die vielen Anregungen und Hinweise, die zum Gelingen dieser Arbeit beigetragen haben.
Auch gilt mein Dank allen Freunden und Bekannten, die mich in irgendeiner Form bei der Anfertigung dieser Diplomarbeit unterstützt haben.
Im Weiteren bedanke ich mich bei meiner Familie, die mich während des gesamten Studiums unterstützt und mir über kleine Durststrecken hinweggeholfen hat.

Mittweida, den 29. Januar 2010 Peter Großöhme

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AEC Availability Environment Classification

AMD-V AMD Virtualization

API Application Programming Interface

BIND Berkeley Internet Name Domain

HA High Availability

ECC Error Correction Code

CCM Consensus Cluster Membership

CIB Cluster Information Base

CM Cluster Manager

CLVM Cluster Logical Volume Manager

CRM Cluster Resource Manager

DLM Distributed Lock Manager

DNS Domain Name System

DRBD Distributed Replicated Block Device

GPL GNU General Public License

GUI Graphical User Interface

HDD Hard Disk Drive

HRG Harvard Research Group

HSRP Hot Standby Router Protocol

HV Hypervisor

IP Internet Protocol

IPMI Intelligent Platform Management Interface

IEEE Institute of Electrical and Electronics Engineers

IVT Intel Virtualization Technology

KISS Keep it simple, stupid

KVM Kernel-based Virtual Machine

LB Load Balancer

LSB Linux Standard Base

LVS Linux Virtual Server

LRM Local Resource Manager

NDB Network Data Base

NMS Netzwerk Management System

NOP No Operation

MGM Management

MMU Memory Management Unit

MPI Message Passing Interface

MTBF Mean Time Between Failures

MTTR Mean Time To Recover

OCF Open Cluster Framework

OCFS2 Oracle Cluster File System 2

OS Operating System

PE Policy Engine

RAID Redundant Array of Independent Disks

RAM Random Access Memory

SCI Scalable Coherent Interface

SLA Service Level Agreement

SLB Server Load Balancing

SNMP Simple Network Management Protocol

SPOF Single Point of Failure

STP Spanning Tree Protocol

UML UserMode Linux

URI Uniform Resource Identifier

USV Unterbrechungsfreie Stromversorgung

VE Virtual Environment

VIP Virtuelle IP

VM Virtuelle Maschine

VMI Virtual Machine Interface

VMM Virtual Machine Monitor

VRRP Virtual Router Redundancy Protocol

WAN Wide Area Network

XML Extensible Markup Language

Kapitel 1

Einleitung

1.1 Aufgabenstellung und Themengebiet

Die Aufgabenstellung umfasst die Analyse von Hochverfügbarkeitsmechanismen in Verbindung mit virtuellen Maschinen. Kombiniert man dabei zielgerichtet verschiedene Optionen der Hardware- sowie Applikationsebene mit den Technologien der Virtualisierung, so lassen sich dadurch hochverfügbare Dienste realisieren, welche auch ökonomischen Randbedingungen genügen.

Im Anschluss an die theoretischen Vorbetrachtungen über die Vereinbarkeit der beiden Konzepte Hochverfügbarkeit und Virtualisierung ist eine geeignete Clusterlösung auszuwählen, mit der sich diese Kombination praktisch umsetzen lässt. Als Voraussetzung für die Auswahl sind bestimmte Mindestanforderungen einzuhalten sowie der Funktionsumfang und die Architektur dieser ausführlich zu beschreiben.

Die Eignung dieser Cluster Suite als Grundlage verschiedener Serverdienste ist durch praktisch umgesetzte Lösungen unter Einhaltung der vorgegebenen Rahmenbedingungen nachzuweisen. Dazu sind, im Sinne der Nachvollziehbarkeit, auf das Wesentliche beschränkte Bespielkonfigurationen zu erstellen. Es soll hierbei auf potentielle Fehlerquellen und auftretende Probleme eingegangen und unterschiedliche Lösungsansätze diskutiert werden. Die gewählten Beispiele sollen auf vergleichbare Dienste übertragbar sein.

Die Arbeit ist im Themengebiet der Technischen Informatik angesiedelt. Ein Teilgebiet davon ist die Rechnerkommunikation. Diese ist wiederum Voraussetzung für die Realisierung von verteilten Systemen, welche hier im Fokus stehen. Für das bessere Verständnis der vorliegenden Diplomarbeit werden Grundlagen der Hardwarearchitektur, Rechnernetze sowie Datenbanken vorausgesetzt.

1.2 Motivation und Ziel der Arbeit

Die bisher weit verbreite Vorgehensweise des herkömmlichen Clusterns ist immer seltener anzutreffen. Vielmehr geht der Trend dahin, die Anzahl physischer Maschinen im Zuge der Virtualisierung zu reduzieren, da die Leistung eines einzelnen physischen Hosts mittlerweile ausreicht, um mehrere Serverdienste parallel anzubieten. Deshalb ist das Betrachten konkreter Hochverfügbarkeitskonfigurationen auf Basis dieses Konzeptes von besonderem Interesse, da sich Virtualisierung und Hochverfügbarkeit im Normalfall ausschließen und einen Widerspruch darstellen.

Virtualisierung senkt die Verfügbarkeit eines Systems, was jedoch durch das geschickte Verteilen der einzelnen Clusterbestandteile kompensiert werden kann. Dabei können viele verschiedene Dienste auf mehreren physischen Maschinen so verteilt werden, dass jeder einzelne Dienst im Fehlerfall dennoch verfügbar bleibt. Dadurch kann weiterhin eine bessere Auslastung der einzelnen Maschinen erreicht werden, was ohne die Fähigkeiten der Virtualisierung nicht denkbar wäre.

Als Ergebnis der Diplomarbeit soll untersucht werden, in welchen Bereichen Hochverfügbarkeit auf Basis virtueller Maschinen ohne unverhältnismäßig hohen Aufwand umgesetzt werden kann. Ziel ist es letztendlich, die vertraglich vereinbarten Anforderungen des Kunden mit möglichst geringem administrativen Aufwand durch einen hohen Automatisierungsgrad zu realisieren.

Die Schwierigkeit besteht darin, diese Mechanismen auf Virtualisierungs- und Applikationsebene so zu kombinieren, ohne dabei die Komplexität in einem solchen Ausmaß zu steigern, dass das Gesamtsystem unkontrollierbar wird. Es soll stattdessen soweit wie möglich unbeaufsichtigt betrieben werden können. Hierbei ist genau zu definieren, welcher Teil der Hochverfügbarkeitslösung in welchem Fehlerfall zu reagieren hat. Die Entscheidung, ob dieser Teil auf Virtualisierungs- oder Applikationsebene realisiert werden soll, ist sorgfältig abzuwägen.

1.3 Kapitelübersicht

Kapitel 2

In erstem Teilabschnitt dieses Kapitels werden Hintergründe zu Ausfallursachen und Möglichkeiten zur Bestimmung einer Verfügbarkeitsstrategie aufgezeigt. Dazu werden u. a. die Begriffe Verfügbarkeit, Hochverfügbarkeit, Fehlertoleranz sowie Single Point of Failure erklärt. Im zweiten Teilabschnitt erfolgt ein Überblick über die Virtualisierungstechnologien. Hierzu werden Virtualisierungsarten detailliert beschrieben und deren Vor- und Nachteile betrachtet. Im Anschluss wird erörtert, ob sich Hochverfügbarkeit mit Virtualisierung in Verbindung bringen lässt. Der letzte Teilabschnitt beschäftigt sich mit Clusterarten, insbesondere der Hochverfügbarkeits-Cluster, und stellt diese kurz dar. Es werden weiterhin Begriffe geklärt, die zur Konfiguration und zum Betrieb eines Clusters von besonderer Wichtigkeit sind.

Kapitel 3

Dieses Kapitel fasst allgemeine Anforderungen an hochverfügbare Systeme auf verschiedenen Ebenen zusammen. Anschließend wird die freie Clusterlösung Linux-HA auf Applikationsebene behandelt, die auf allen gängigen Linux-Distributionen eingesetzt werden kann. Im Verlauf der Beschreibung dieser Cluster Suite wird zunächst ein Überblick gegeben, um folgend die einzelnen Versionen mit ihren Funktionalitäten und Komponenten zu skizzieren. Ergänzend dazu finden sich im letzten Abschnitt dieses Kapitels eine Reihe ausgewählter Linux-HA Partnerprojekte, um die Hochverfügbarkeitslösungen abzurunden.

Kapitel 4

Basierend auf den in Kapitel 3 vorgestellten Implementierungen von Hochverfügbarkeitsmechanismen werden in diesem Kapitel drei grundlegende praxisnahe Beispiele detailliert betrachtet. Dabei handelt es sich um die häufig benötigten Dienste eines hochverfügbaren File-, Web- sowie Datenbankservers. Zu jedem dieser Beispiele erfolgt jeweils ein kurzer Themenabriss mit anschließender Zusammenfassung der wichtigsten Installations- und Konfigurationsschritte. Schwerpunkt sind die am häufigsten auftretenden Probleme und die entsprechenden Lösungsstrategien werden diskutiert.

Kapitel 5

Das letzte Kapitel schätzt die erreichten Ergebnisse in Bezug auf die Aufgabenstellung ein und fasst die wichtigsten Erkenntnisse zusammen. Außerdem wird auf mögliche Verbesserungen sowie alternative Lösungsansätze hingewiesen.

Kapitel 2

Theoretische Grundlagen

2.1 Hochverfügbarkeit

2.1.1 Begriffsbestimmung und Definition

[Hel04] [Bre07]

Ein Dienst gilt als verfügbar, solange dieser die Aufgaben erfüllt, für die er vorgesehen ist. Als Verfügbarkeit bezeichnet man die Wahrscheinlichkeit, dass ein System für einen spezifizierten Zeitraum funktionstüchtig bzw. verfügbar ist. Das Verhältnis aus Downtime und Uptime ergibt die Verfügbarkeit:

$$\text{Verfügbarkeit} = \frac{\text{Uptime}}{\text{Downtime+Uptime}}$$

Ein System wird als hochverfügbar bezeichnet, sobald eine Anwendung auch im Fehlerfall weiterhin verfügbar ist und ohne direkten menschlichen Eingriff weitergenutzt werden kann. Dies bedeutet, dass der Anwender keine oder nur eine kurze Unterbrechung wahrnimmt. Hochverfügbarkeit (HA - abgeleitet vom engl. High Availability) bezeichnet damit die Fähigkeit eines Systems, auch bei Ausfall einzelner Komponenten einen uneingeschränkten Betrieb zu gewährleisten.

Das Institute of Electrical and Electronics Engineers (IEEE) definiert den Begriff High Availability wie folgt:

> "High Availability (HA for short) refers to the availability of resources in a computer system, in the wake of component failures in the system." [CC04]

Die Havard Research Group bezieht den Begriff Hochverfügbarkeit auf den prozentualen Wert der Verfügbarkeit, diese wird in Verfügbarkeitsklassen (siehe auch Tabelle AEC-

Klassen 2.1.4 auf Seite 12) eingeteilt [HRG01]. Nach dieser Klassifizierung gilt ein System ab einer Verfügbarkeit von mindestens 99,99 Prozent als hochverfügbar.

Die Erhöhung der Verfügbarkeit bis hin zur Hochverfügbarkeit wird im Regelfall durch die Verringerung von Downtimes realisiert, diese können, müssen aber nicht, geplant sein. Geplante Downtimes sind z. B. Wartungsfenster für Hardware- und Softwareupgrades, ungeplante dagegen resultieren meist aus Fehlern. Letztere werden im Rahmen hochverfügbarer Systeme durch fehlertolerante Hard- und Software sowie durch Vermeidung von Single Point of Failures (SPOF) so aufgefangen, dass das System auch im Fehlerfall weiterhin verfügbar bleibt.

HA-Architekturen verfügen meist über folgende Eigenschaften [Hel04]:

- Toleranz und Transparenz gegenüber Fehlern
- präventive Build In-Funktionalitäten
- proaktives Monitoring und schnelle Fehlererkennung
- schnelle Wiederherstellungsmöglichkeiten
- automatisierte Wiederherstellung ohne administrative Eingriffe
- kein oder geringer Datenverlust
  (z. B. Rollback einer Datenbank bei Verlust der Session)

Die Zuverlässigkeit (Realiability) hingegen ist eine Eigenschaft, die angibt, wie verlässlich von einem Produkt eine zugewiesene Aufgabe in einem Zeitintervall erfüllt wird. Diese wird oft mit der Verfügbarkeit verwechselt, ist aber nicht direkt messbar und wird entweder empirisch durch Beobachtung der Ausfallhäufigkeit ermittelt oder analytisch aus der Zuverlässigkeit der Einzelkomponenten bestimmt. Ein System mit einer geringen Zuverlässigkeit kann immer noch hochverfügbar sein, wenn die Zeit zur Wiederinbetriebnahme in einem Fehlerfall sehr kurz ist. Das ist zum Beispiel der Fall, sobald ein Dienst jeden Monat einmal ausfällt, aber schon nach einer Minute wieder verfügbar ist. In diesem Szenario wird bereits eine Verfügbarkeit von mehr als 99,99% über das Jahr erreicht. Letztendlich hängen die genauen Anforderungen für die Verfügbarkeit eines Dienstes vom jeweiligen Einsatzgebiet ab. Deshalb muss jede Konfiguration individuell geprüft werden, inwieweit sich Ausfälle, auch von kurzer Dauer, als problematisch erweisen können.

Trotz allem müssen in hochverfügbaren Systemen geplante und ungeplante Ausfallzeiten auf ein Minimum reduziert werden. Um geeignete Maßnahmen treffen zu können, müssen Systemausfälle und deren Gründe kategorisiert werden. Im folgenden Kapitel 2.1.2 werden diese näher betrachtet.

2.1.2 Ursachen für Systemausfälle

[Micb] [Hel04]

Die Nichtverfügbarkeit eines Dienstes wird als Systemausfall bezeichnet. Die Ursachen für Systemausfälle sind weit gestreut und setzen die Verfügbarkeit eines System meist stark herab. Man unterteilt diese in geplante und ungeplante Ausfälle. Als ungeplant werden Ausfallzeiten bezeichnet, die in Folge eines Fehlers aufgetreten sind bzw. in der das System aufgrund von unvorhersehbaren Ereignissen nicht verfügbar ist. Hingegen werden Ausfallzeiten, die durch das Herunterfahren eines Systems durch den Administrator zu einer festgelegten Zeit erfolgen, als geplant betrachtet. Die Zeit sollte möglichst so gewählt werden, dass dadurch die Produktivität eines Unternehmens gar nicht bzw. nur gering beeinflusst wird. Nachfolgend eine Tabelle mit möglichen Ursachen für Systemausfälle:

Ursache für Ausfallzeiten	**Beispiele**
vom Administrator geplante Ausfallzeit	Aktualisierung von Hardwarekomponenten, Firmware, Treibern, Betriebssystem oder Softwareanwendungen
Komponentenfehler	fehlerhafte Serverkomponenten, z. B. Speicherchips, Lüfter, Systemkarten oder Stromversorgung; fehlerhafte Speicherkomponenten in untergeordneten Systemen, wie ausgefallene Laufwerke oder Laufwerkscontroller; fehlerhafte Netzwerkkomponenten, z. B. Router oder Netzwerkkabel
Softwarefehler oder -ausfälle	keine Reaktion des Laufwerks, keine Reaktion oder Neustart des Betriebssystems, Viren, beschädigte Dateien
fehlerhafte Bedienung oder böswillige Benutzer	versehentliches oder absichtliches Löschen von Dateien, Bedienfehler
Systemausfall oder -wartung	erforderlicher Neustart von Software oder von Systemen oder Ausfall der Systemplatine
Lokaler Notfall	Brände, schwere Stürme oder andere lokal begrenzte Ereignisse
Regionaler Notfall	Erdbeben, Orkane, Überschwemmungen oder regionale Naturkatastrophen

Tabelle 2.1: Ursachen für Ausfallzeiten und Fallbeispiele [Micb]

Damit lassen sich im Allgemeinen folgende Fehlertypen kategorisieren:

- Komponentenfehler
- Speicherfehler (Storage)
- Netzwerkfehler
- Softwarefehler
- Standortfehler

Insbesondere ungeplante Systemausfälle können nicht vollständig vermieden werden, jedoch durch entsprechende Vorkehrungen erheblich reduziert werden. Hochverfügbarkeit erfordert dennoch die Vermeidung bzw. Beschränkung der ungeplanten Ausfallzeiten.

2.1.3 Metriken

[Hel04] [Sch09] [Bre07]

Kennzahlen

Die Verfügbarkeit eines Systems wird u. a. mittels folgender Kenngrößen bewertet:

- **Zuverlässigkeit**
  mittlere Uptime des Systems (Erfüllung der Funktionstüchtigkeit)
- **Reaktionszeit**
  Dauer, bis ein System eine spezielle Aktion ausführt
- **Mean Time Between Failures** (MTBF)
  mittlere Betriebsdauer zwischen zwei Ausfällen
- **Mean Time To Recover** (MTTR)
  mittlere Zeit zur Wiederherstellung nach einem Ausfall
- **Verfügbarkeitsklasse**
  (siehe auch Tabelle 2.1.4 auf Seite 12)
- **maximale Dauer eines Ausfalles**

Da oftmals bei Berechnung der genannten Kenngrößen nur die Hardware in Betracht gezogen wird, jedoch die Anforderung der HA für ein gesamtes System inkl. aller Applikationen gilt, müssen alle Bestandteile berücksichtigt werden. Sobald die Störung durch einen versehentlichen Benutzereingriff herbeigeführt ist, nützt an dieser Stelle auch keine zuverlässige fehlertolerante Hardware. Aus diesem Grund muss das Gesamtsystem einschließlich aller beteiligten Komponenten betrachtet werden. Ferner darf keine dieser Komponenten einen SPOF aufweisen. Des Weiteren sollten diese fehlertolerant reagieren können.

Mathematische Grundlagen

Für die Verfügbarkeit V gilt:

$$\text{Verfügbarkeit } V = \frac{\text{MTBF}}{\text{MTBF+MTTR}}$$

Für die Nichtverfügbarkeit N gilt:

$$\text{Nichtverfügbarkeit } N = \frac{\text{MTTR}}{\text{MTBF+MTTR}}$$

Daraus resultiert für die MTTR:

$$\text{MTTR} = \text{MTBF} * \frac{1 - V}{V}$$

Die Ausfallrate λ ergibt sich aus:

$$\lambda = \frac{1}{\text{MTBF}}$$

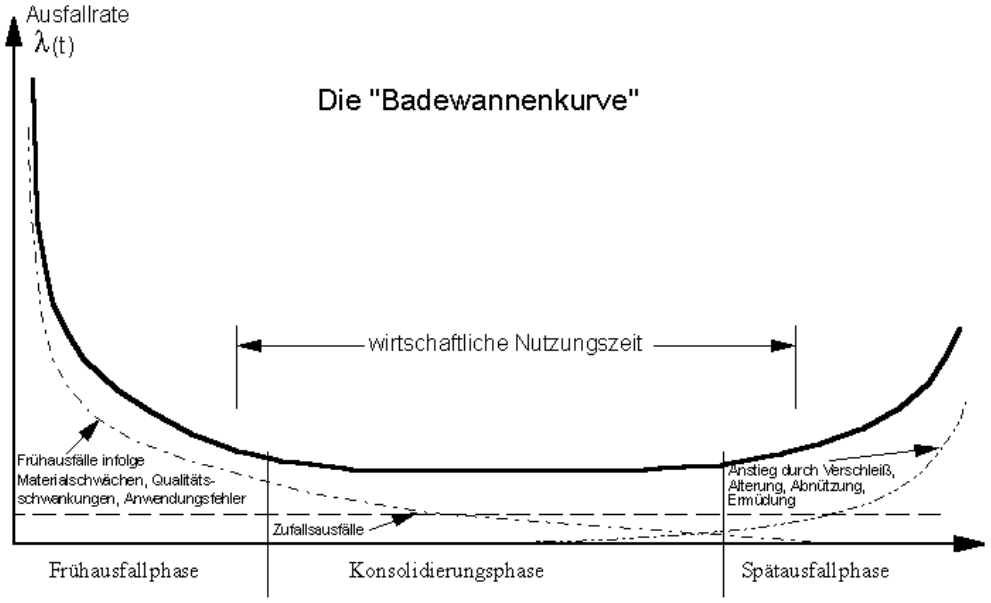

Abbildung 2.1: Typischer Verlauf der Ausfallrate - Badewannenkurve [Bec05]

Ausfälle sind zeitlich nicht gleich verteilt. Das heißt im Detail, dass die Zeiträume zwischen zwei Fehlern ungleich sind. Unmittelbar nach der Neuanschaffung einer Komponente ist die Ausfallwahrscheinlichkeit sehr hoch. Man spricht hier von sogenannten Frühausfällen. Verursacht werden diese durch allgemeine Konstruktions- und Designmängel der jeweiligen Komponente, Fehler in der Produktion sowie durch Installations- und Transportfehler. Beim Abschluss der Produktion werden Komponenten teilweise einem künstlichen Alterungsprozess unterzogen, um Frühausfälle vor Auslieferung auszusortieren. Im Anschluss daran folgt die Phase der statistischen Ausfälle, die durch die Gleichung der Ausfallrate λ beschrieben wird. Diese ist annähernd konstant. Die festgelegte Nutzungsdauer, für die eine Komponente produziert wurde, endet in der Regel am Ende dieses Zeitabschnitts. Mit zunehmendem Alter kommt es durch Alterung und Verschleiß erneut zu verstärkten Ausfällen. Ein rechtzeitiger Austausch zu Beginn dieser Phase beugt dem Ausfall vor. Der Ausfallverteilung liegt die Badewannenkurve (siehe Abbildung 2.1) zugrunde.

Systemverfügbarkeit

Das Zusammenspiel vieler Einzelkomponenten bestimmt letztendlich die Systemverfügbarkeit, damit ein Server auch einen Dienst erbringen kann. Die Stromversorgung ist für die Grundversorgung ein wichtiger Aspekt und trägt zur wesentlichen Voraussetzung für die Funktionsfähigkeit eines Dienstes bei. Wie in [Sch09] beschrieben, beträgt die durchschnittliche Ausfallzeit der Stromversorgung in Deutschland 23 Minuten, was einer Verfügbarkeit von 99,996% im Jahr entspricht. Wer also einen hochverfügbaren Dienst mit mehr als 99,996% anbieten möchte, kommt um die Planung einer unterbrechungsfreien Stromversorgung nicht herum. Das ist natürlich auch im Weiteren sinnvoll, da ein Stromausfall bei einem Serversystem unter Umständen Beschädigungen zur Folge hat. Alle flüchtig gespeicherten Daten, z. B. im RAM oder nicht batteriegepufferten Caches, sind unwiderruflich verloren. Außerdem erhöht ein Stillstand der Festplatten nach einem Dauerbetrieb (24/7) die Gefahr eines Ausfalls bei Wiederinbetriebnahme des Systems. Die Stromversorgung ist aber nur ein Glied in der Kette, um eine einwandfreie Funktion sicherzustellen.

Darüber hinaus ist die Funktionalität sämtlicher Serverhardware, wie z. B. Motherboard, Prozessor oder RAM, erforderlich, damit ein Dienst auf einem Server laufen kann. Zum Datentransport müssen ferner alle beteiligten Netzwerkkomponenten, wie z. B. Router, Switches und Kabel, in Ordnung sein. Anhand der genannten Einflussfaktoren ist es grundsätzlich möglich eine Formel für die Systemverfügbarkeit herzuleiten, indem man komplexe zusammengesetzte Systeme auf einfache Bausteine in Einzelkomponenten überführt und diese auf zwei unterschiedliche Arten miteinander verschaltet:

- **Serielle Kopplung (Reihenschaltung)**
  Ein seriell gekoppeltes System besteht aus zwei Komponenten. Das Funktionieren des Gesamtsystems hängt in jedem Fall von der Funktion beider Komponenten ab. Daraus ergibt sich für die Verfügbarkeit des Gesamtsystems V_{ser} aus dem Produkt der Einzelverfügbarkeiten aller Komponenten wie folgt:

  $$V_{\text{ser}} = V_1 * V_2$$

  Die Gesamtverfügbarkeit ist immer kleiner als die kleinste Verfügbarkeit der einzelnen Komponenten. Möchte man die Gesamtverfügbarkeit eines Systems verbessern, so muss immer die Komponente mit der geringeren Einzelverfügbarkeit verbessert oder gegen eine andere ausgetauscht werden. Die Fehlerquote eines Systems steigt mit zunehmender Anzahl der in Serie geschalteten Komponenten weiter an.
  Es gilt: Eine Kette ist immer nur so stark wie ihr schwächstes Glied.

- **Parallele Kopplung und Redundanz (Parallelschaltung)**
  Ein parallele Kopplung liegt vor, wenn zwei Komponenten die gleiche Aufgabe erfüllen, d. h. redundant ausgelegt sind. Im Gegensatz zur seriellen Kopplung hängt die Funktion des Gesamtsystems hingegen nur von einer der beiden Komponenten ab - die Komponenten sind unabhängig voneinander. Die Verfügbarkeit V_{par} resultiert aus eins minus der Wahrscheinlichkeit des Ausfalls aller Komponenten wie folgt:

  $$V_{\text{par}} = 1 - (1 - V_1) * (1 - V_2)$$

  Falls beide Komponenten eine identische Verfügbarkeit aufweisen, reduziert sich die Formel auf:

  $$V_{\text{par}} = 1 - (1 - V_1)^2$$

  Die Verfügbarkeit des Gesamtsystems ist in diesem Fall immer höher als die Einzelverfügbarkeit der jeweiligen Komponenten. Die Redundanz ist ein Maß dafür, wie oft eine einzelne Komponente im System vorhanden ist.

Aufgrund dieser Kopplungsmöglichkeiten folgt die Grundlage der 3R-Regel für hochverfügbare Systeme:

> "Redundanz, Redundanz und noch einmal Redundanz." [Sch09]

Dieser Grundsatz sollte auch bei einer Anforderungsplanung zur Konzeption eines hochverfügbaren Clustersystems berücksichtigt werden. Für die Verfügbarkeit der einzelnen Bausteine kann keine 100%ige Berechnung vorgenommen werden. Es ist dennoch hilfreich, die kritischen Komponenten ausfindig gemacht zu haben und auf Basis der ermittelten Verfügbarkeit den Grad der Redundanz zu bestimmen. Dadurch werden spätere Schwierigkeiten verhindert. Die hohe Kunst der Hochverfügbarkeit ist ein Design, bei dem SPOF durch den Einsatz von redundanten Einzelkomponenten vermieden werden.

Man sollte aber auch bedenken, dass mit steigender Redundanz der Komponenten sich die Komplexität des Gesamtsystems erhöht, was wiederum eine potentielle Fehlerquelle darstellt. Interessant wäre hierbei die Betrachtung, ob man die Redundanz theoretisch soweit auslegen kann, dass sich dadurch die Gesamtverfügbarkeit aufgrund der Komplexität wieder verringert, ohne dabei die entstehenden Kosten zu berücksichtigen. Deshalb muss im Einzelfall geprüft werden, ob nicht eine weniger redundante, aber dafür robustere Komponente für den Einsatz sinnvoller scheint.

2.1.4 Verfügbarkeitsklassen

[Hel04]

Die Verfügbarkeit von Computersystemen wird meist in Dauer der Downtime pro Jahr gemessen und in Prozent angegeben.

Die Havard Research Group (HRG) teilt Hochverfügbarkeit in ihrer Availability Environment Classification (AEC) in sechs Klassen ein [HRG01]:

- Conventional (AEC-0): Funktion kann unterbrochen werden, Datenintegrität ist nicht essenziell.
- Highly Reliable (AEC-1): Funktion kann unterbrochen werden, Datenintegrität muss jedoch gewährleistet sein.
- High Availability (AEC-2): Funktion darf nur innerhalb festgelegter Zeiten oder zur Hauptbetriebszeit minimal unterbrochen werden.
- Fault Resilient (AEC-3): Funktion muss innerhalb festgelegter Zeiten oder während der Hauptbetriebszeit ununterbrochen aufrechterhalten werden.
- Fault Tolerant (AEC-4): Funktion muss ununterbrochen aufrechterhalten werden, 24/7-Betrieb (24 Stunden, 7 Tage die Woche) muss gewährleistet sein.
- Disaster Tolerant (AEC-5): Funktion muss unter allen Umständen verfügbar sein.

Verfügbarkeitsklasse	**Bezeichnung**	**Verfügbarkeit**	**Downtime pro Jahr**
2 $\widehat{=}$ AEC-0	stabil	99,0%	3,7 Tage
3 $\widehat{=}$ AEC-1	verfügbar	99,9%	8,8 Stunden
4 $\widehat{=}$ AEC-2	hochverfügbar	99,99%	52,6 Minuten
5 $\widehat{=}$ AEC-3	fehlerunempfindlich	99,999%	5,3 Minuten
6 $\widehat{=}$ AEC-4	fehlertolerant	99,9999%	32 Sekunden
7 $\widehat{=}$ AEC-5	fehlerresistent	99,99999%	3 Sekunden

Tabelle 2.2: AEC-Verfügbarkeitsklassen nach HRG [Hel04]

Im geschäftlichen Umfeld wird die Verfügbarkeit im Rahmen von Service Level Agreements (SLA) geregelt. Ein SLA ist eine Vereinbarung zwischen einem Dienstleister und dem Endkunden und regelt die Art und den Umfang einer angebotenen Leistung. Besonders im Zusammenhang mit der HA werden Qualität und Verfügbarkeit der angebotenen Leistung festgehalten. Daraus resultieren eventuelle Schadensersatzansprüche gegenüber dem Dienstleister im Falle der Nichteinhaltung der vertraglich zugesicherten Eigenschaften. Für die Virtualisierung von Systemen, speziell bei der Ressourcenzuteilung und -nutzung, sind SLAs von besonderer Wichtigkeit.

2.1.5 Kosten contra Verfügbarkeit

[Hel04]

Bei der Definition einer Hochverfügbarkeitsstrategie ist es notwendig, sowohl die technischen Möglichkeiten als auch die ökonomischen Randbedingungen zu analysieren. Für kritische Geschäftsanwendungen, wie z. B. Lagersysteme in Logistikzentren oder Steuerungssysteme für Fertigungsmaschinen im Schichtbetrieb, ist eine Verfügbarkeit von 99,99% üblich, die nach Einteilung in die HRG-Klassen als hochverfügbar gilt. Die Zeiten, in denen Unternehmen versuchten eine 99,999-prozentige Verfügbarkeit zu erlangen, sind vorüber, da die finanziellen Mittel zur Realisierung einer solchen Zielvorgabe ins Unermessliche ansteigen würden. Diese Aussage ist natürlich immer relativ und muss im Verhältnis zum entstehenden Schaden betrachtet werden. Sobald ein Server einen Ausfall in Millionenhöhe verursacht, ist auch fehlertolerante Hardware im Gegensatz dazu noch bezahlbar. [Len09] Unter dem steigenden Zwang zur Kosteneinsparung im IT-Bereich orientiert man sich wieder verstärkt an den Erfordernissen des Geschäftszweiges, indem eine Risikobewertung durchführt und nicht mehr die Verfügbarkeit realisiert wird, die maximal möglich ist.

Vorerst sollte man sich die Frage stellen, welche Kosten (z. B. Produktionsausfall) und auch Schadensersatzansprüche im Falle der Nichtverfügbarkeit eines Dienstes entstehen. Dabei sollten auch die Kosten für verschieden lange Ausfallzeiten betrachtet werden, da diese mit fortschreitender Zeit meist stufenweise ansteigen (siehe Abbildung 2.2).

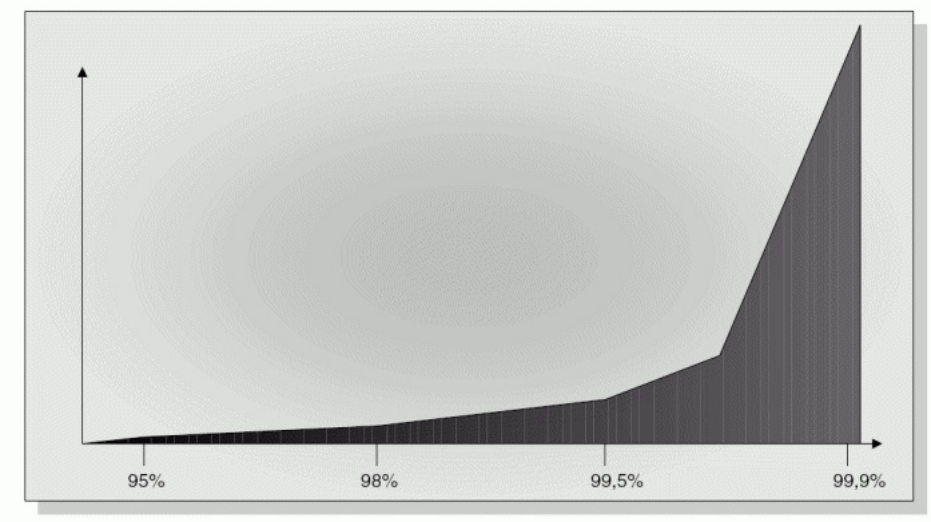

Abbildung 2.2: Kostensteigerungskurve nach Verfügbarkeit [Hel04]

Darauf basierend sollte ein ausreichendes Konzept für eine Notfallwiederherstellung, auch als Disaster Recovery bezeichnet, erstellt werden. Als Grundlage dafür dient das Risi-

komanagement. Es hilft das Auftreten von Notfallsituationen zu vermeiden bzw. deren Auswirkungen zu begrenzen. Mit diesem Wissen lassen sich erforderliche und geeignete Gegenmaßnahmen planen und ergreifen. Bei einer Analyse der Ausfallkosten müssen im Detail u. a. folgende Faktoren berücksichtigt werden:

- betroffene Anwendungen
- Einnahmeverluste je Ausfallminute
- Produktionsausfälle je Ausfallminute
- Kosten für Personal, Räumlichkeiten, usw. je Ausfallminute
- Länge der Ausfallzeit
- Verlust von Kunden
- Rechtskosten und Schadensersatzforderungen von Kunden und Geschäftspartnern
- möglicher Datenverlust und dessen Auswirkung

Zur Ermittlung der anfallenden Kosten findet man im Internet einige Downtime-Kalkulatoren (siehe Anhang A.1), die bei Berechnung für Systemausfälle zur Unterstützung herangezogen werden können. Dazu ist es notwendig, einige Eckdaten des Geschäftsbereiches zu kennen. Neben Personalkosten spielen zur Berechnung der Ausfallkosten auch die Geschäftszeiten und mittlere zu erwartende Einnahmen eine wichtige Rolle. Auf Grundlage dieser sollten die Kosten zur Realisierung einer Verfügbarkeitslösung den Ausfallkosten gegenübergestellt werden, d. h. Kosten und Risiko gegeneinander abwägen. Erst dadurch können vernünftige Entscheidungen getroffen werden, in welcher Art und in welchem Umfang die Verfügbarkeitsstrategie realistisch geplant und umgesetzt werden kann.

Zur abschließenden Bewertung einer Disaster Recovery-Strategie gibt es die Bedarfsanalyse. Diese gliedert sich in Betrachtungen des Bestandes, möglicher Gefahren für die IT-Umgebung, der durch Ausfälle verursachten Risiken sowie Kosten. Die Bedarfsanalyse umfasst die folgenden Teilschritte:

- Bestandsanalyse
- Gefahrenanalyse
- Risikoanalyse
- Kostenanalyse
- Break-Even-Analyse

Weitere Informationen zur Evaluierung eines Disaster-Recovery-Konzepts in [Hel04].

2.2 Virtualisierung

2.2.1 Einführung und Definition

[Pic09] [SBZD07]

Mittlerweile ist es in der Informatik nicht mehr möglich, eine eindeutige Definition des Begriffs der Virtualisierung zu geben, da zu viele Anwendungsfälle (auch außerhalb der Informatik) und Konzepte existieren. Im Allgemeinen ermöglicht Virtualisierung den Betrieb mehrerer Betriebssysteme in einer wohldefinierten Hardwareumgebung, unabhängig davon, welche Hardware im physischen System verbaut ist. Ausgenommen hiervon sind Sonderformen der Virtualisierung, die nachfolgend detailliert betrachtet werden. Ziel der Virtualisierung ist letztendlich die effizientere Nutzung der Ressourcen eines Computersystems, insbesondere im Serverbereich.

Genauer betrachtet gehen die Ansätze der Virtualisierung bis in die zweite Hälfte des 20. Jahrhunderts zurück. Bereits 1974 beschäftigten sich Gerald Popek und Robert Goldberg in [PG74] mit den Anforderungen, die für eine Virtualisierung erforderlich sind. Hierbei stellten sie folgende Ansprüche an ein System:

- Gleichheit: Jedes Programm in einer virtuellen Umgebung muss sich genauso wie auf nativer Hardware verhalten.
- Ressourcenverwaltung: Der Virtual Machine Manager (VMM) muss die komplette Kontrolle über die Ressourcen erhalten.
- Effizienz: Ein Großteil des Instruktionscodes des virtuellen Prozesses muss direkt auf dem physischen Prozessor, ohne Eingreifen des VMM, ausgeführt werden.

Im Jahre 1967 hat IBM den Vorläufer von virtuellen Maschinen und Mainframes entwickelt. Der entwickelte Hypervisor ist eine Technologie, die später auf dem Mainframe als VM populär wurde und heutzutage unter dem Namen z/VM bekannt ist. Bis die Virtualisierung den Durchbruch im Servermarkt schaffte, vergingen jedoch noch einige Jahre.

Letztendlich wurde im Jahre 1998 VMware Global, Inc. gegründet. Bereits im darauffolgenden Jahr sind erste Virtualisierungslösungen verkauft worden. Ursprünglich entstand das Unternehmen aus einem Forschungsprojekt der Universität Stanford. Auch heute noch ist VMware Marktführer mit den angebotenen Virtualisierungslösungen und arbeitet zum Teil eng mit den Prozessorherstellern zusammen, um die Virtualisierung stetig zu verbessern und auch voranzutreiben. In dieser Zeit entstanden auch eine Reihe von freien Virtualisierungslösungen. Die wohl populärste ist XEN, ein Projekt der University of Cambridge, welches erstmalig 2003 erschienen ist.

2.2.2 Virtualisierungsarten

Full Virtualization

[SBZD07]

Bei der Vollvirtualisierung ist das Gastbetriebssystem ohne vorherige Anpassung (Kernel) in der virtuellen Umgebung lauffähig. Es hat in diesem Fall keine Kenntnis davon, dass es sich in einer virtuellen Umgebung befindet und verhält sich wie jedes andere laufende Betriebssystem. Das hat den Vorteil, dass man nicht auf ein bestimmtes Betriebssystem beschränkt ist. Dies führt jedoch dazu, dass die Hardware nahezu vollständig virtualisiert werden muss, da ein Zugriff auf die physikalische Hardware nicht möglich ist. Dadurch wird eine einheitliche Standardhardware für jedes Gastbetriebssystem geschaffen, meist mit Geräten, bei denen die Treiber gleich vom jeweiligen Betriebssystemhersteller mitgeliefert werden. Ferner kann man dadurch die virtuellen Maschinen auf andere physische Hardware verschieben (Migration), ohne dass vorherige Anpassungen an der Treiberkonfiguration notwendig sind.

Sobald die virtuelle Maschine gestartet wird, sieht das Gastsystem dabei eine virtuelle physische Maschine mit BIOS, Prozessor und sonstiger Hardware - zusätzliche Steckkarten werden nicht durchgereicht. Vor der virtuellen Umgebung befindet sich der Virtual Machine Monitor (VMM). Die Aufgabe des VMM besteht darin, die Hardwarezugriffe der VM zu überwachen und an die physische Hardware weiterzureichen. Wie man bereits hier erkennen kann, ist dies unter Umständen sehr aufwändig und führt zu Performanceeinbußen, welche einen Nachteil der Vollvirtualisierung darstellt. Bei vielen Virtualisierungslösungen setzt der VMM auf ein Hostbetriebssystem auf.

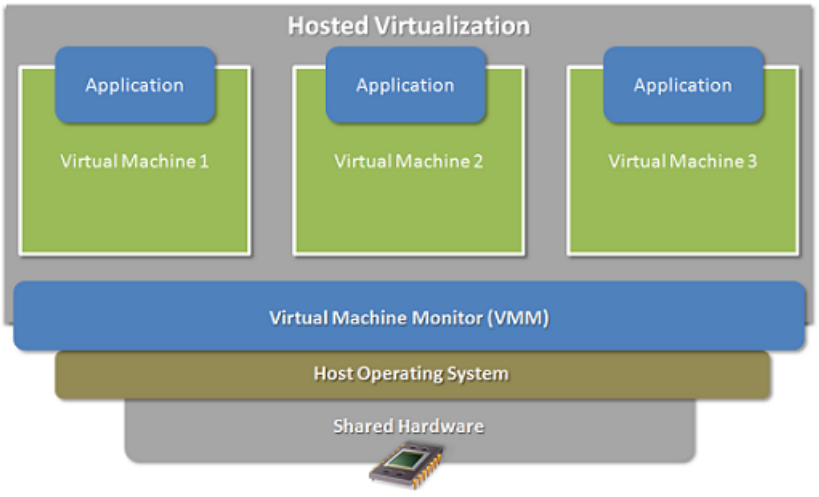

Abbildung 2.3: Full Virtualization: Hosted [NI09]

Eine Ausnahme stellt hierbei VMwares ESX-Server dar, welcher den VMM in Form eines Betriebssystem-Kernels implementiert, der direkten Zugriff auf die Hardware hat. Da VMware dieses Produkt speziell für den Einsatz im Serverbereich entwickelt hat, steht die Performance mit an erster Stelle. Dafür mussten sämtliche Hardwaretreiber und Userspace-Tools für den ESX-Server neu entwickelt werden, was das Spektrum an unterstützter Hardware stark einschränkt.

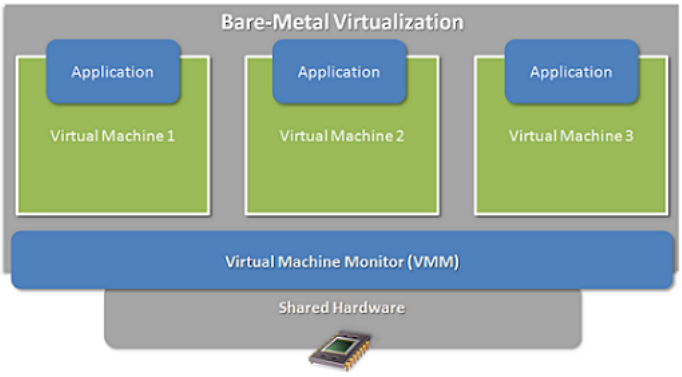

Abbildung 2.4: Full Virtualization: Bare-Metal [NI09]

VMware profitiert dabei nicht von der ursprünglichen Hardwarevirtualisierung der Prozessoren, sondern setzt diese komplett in Software um. VMware nutzt stattdessen Binary Translation und Direct Execution. Bei dieser handelt es sich um Ring-3-Zugriffe, welche durch Applikationen im Gastbetriebssystem gemacht werden. Diese werden ohne Eingriff des VMM direkt vom Prozessor ausgeführt, so wie es der Name bereits vermuten lässt. Komplexer hingegen sieht es bei Binary Translation (Patchen von Kernel-Mode Code) aus - hier handelt es sich um Ring-0-Zugriffe (Kernel Code), die das Gastbetriebssystem selbst durchführt. Dabei wird der gesamte Instruktionscode bis zum nächsten Sprung erst gescannt, decodiert und kommt erst dann zur Ausführung, nachdem dieser optimiert und teilweise sogar noch in einen unkritischen Ring-3-Zugriff geändert wurde. Auf diese Weise kann ein aufwändiger Context-Switch vermieden werden. Anzumerken ist dabei noch, dass die Länge der ursprünglichen Codesequenz nicht mehr mit der Länge der Optimierten übereinstimmt. Das macht die Angelegenheit sehr komplex, da sich auch die Adressen der einzelnen Befehle ändern. Sofern keine Optimierung möglich scheint, wird ein kritischer Befehl durch einen Debug Interrupt überschrieben und mit No Operations (NOP) aufgefüllt. Ein weiterer Trick, den VMware anwendet, ist das Zwischenspeichern bereits ausgeführten Codes, so dass keine erneute Bearbeitung durch den Hypervisor notwendig ist. Das bringt massiven Performancegewinn - teilweise hat hier VMware schon Vorarbeit geleistet und liefert bereits übersetzten Code mit, abhängig davon, welches Gastbetriebssystem ausgewählt wurde.

Paravirtualisierung

[SBZD07]

Die Paravirtualisierung trennt host- und gastspezifische Implementierung. Dazu muss das Gastsystem angepasst (Kernel Patch) werden, damit eine definierte Schnittstelle für die Virtualisierung bereitgestellt werden kann. Das Gastsystem weiß damit, dass es virtualisiert ist und reicht alle privilegierten Befehle über diese Schnittstelle direkt an den Hypervisor (HV) zur Bearbeitung weiter. Dadurch ist es nicht notwendig, die VM des Gastsystems zu überwachen und alle kritischen Befehle abzufangen, so wie es bei der Vollvirtualisierung der Fall ist. Es wird nur das Betriebssystem virtuell gestartet, jedoch keine Hardware emuliert. Die Schnittstelle ist letztendlich eine abstrakte Verwaltungsschicht, die ähnlich der, aber nicht identisch zur physisch verbauten Hardware ist.

Die Funktionalität des HV erhält man durch einen Kernel Patch des Hostbetriebssystems, der dadurch zum eigentlichen Bestandteil des Betriebssystemkerns wird. Genau betrachtet sitzt der HV zwischen Hardware und Betriebssystemkern und ist damit sehr hardwarenah konzipiert.

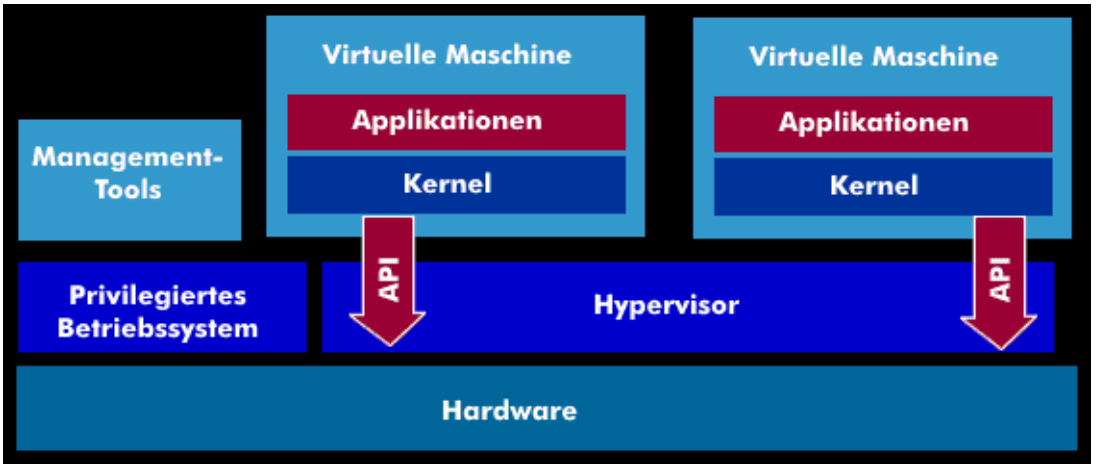

Abbildung 2.5: Paravirtualisierung [ITW]

Damit gilt die Paravirtualisierung als sehr performant und der Verlust beziffert sich auf wenige Prozent im Vergleich zu einem nicht virtualisierten System. Ein Nachteil der Paravirtualisierung stellt die Anpassung des Host- und insbesondere des Gastsystems dar, da es die Auswahl der zu virtualisierenden Betriebssysteme stark einschränkt. Die Vorteile der Paravirtualisierung können besonders dann ausgeschöpft werden, wenn Host- und Gastbetriebssystem vom gleichen Typ sind. Bedacht werden sollte dabei auch, dass beim Umzug einer VM auf andere Hardware (Migration) evtl. Treiberanpassungen erfolgen müssen, da die Hardware weitgehend an die Gäste durchgereicht wird.

Wie bereits in der Einführung angedeutet, ist der klassische und beliebteste Vertreter XEN [Xen]. XEN basiert auf einer Erweiterung des x86-Befehlssatzes und wurde auf Basis des Linux-Kernels entwickelt. Der große Vorteil davon ist, dass der XEN-Host die gesamte Hardware nutzen kann, die bereits vom jeweiligen Linux-Kernel unterstützt wird. XEN ist jedoch nicht an Linux gebunden - es existieren z. B. Varianten für NetBSD oder Sun Solaris. Im Jahre 2007 wurde XenSource Inc. größtenteils vom amerikanischen Softwarehersteller Citrix aufgekauft. Zur Unterstützung wird das Projekt u. a. von vielen bekannten IT-Konzernen, wie z. B. IBM, HP, Intel, AMD, Sun, Microsoft und Novell gepflegt.

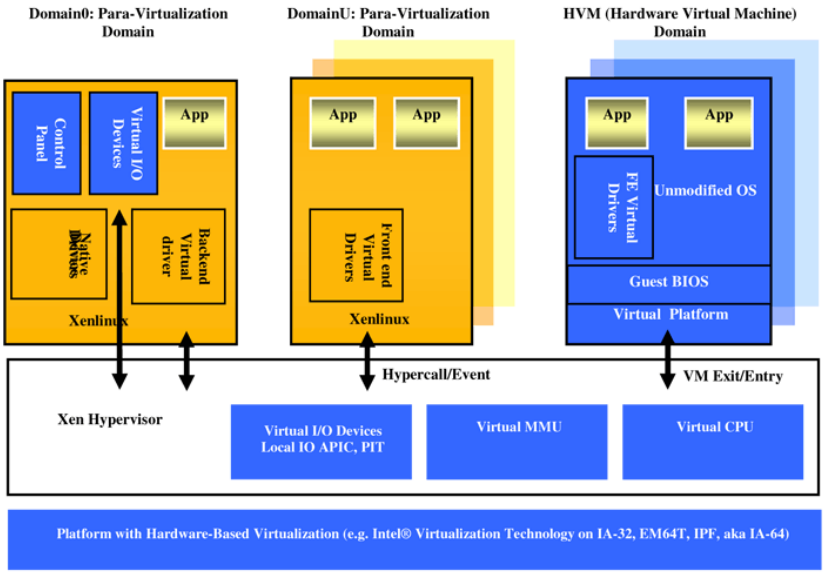

Abbildung 2.6: Paravirtualisierung: XEN mit CPU-Virtualisierung [Int06]

Im Vergleich zu allen anderen Virtualisierungsarten mit Ausnahme von VMware ESX ist die Paravirtualisierung sehr performant. VMware hat dennoch eine offene Schnittstelle für die Paravirtualisierung von Betriebssystemen, insbesondere für Linux, implementiert, welche unter dem Namen Virtual Machine Interface (VMI) bekannt ist [VGc]. Ab Linux-Kernel 2.6.22 wird Paravirtualisierung unterstützt. Diese ist im Menüpunkt *Processor type and features* zu finden, in dem die Unterstützung für die XEN- oder VMI-API aktiviert werden kann. Laut einigen Erfahrungsberichten funktioniert die Paravirtualisierung nicht sonderlich gut mit Microsofts Windows Server 2003/2008. Die Effekte reichen vom Absturz beim Bootvorgang bis hin zu nicht korrekt erkannten Hardware-Komponenten. VMI wird bei eingeschalteter CPU-Virtualisierung für 64-Bit Gäste unter VMware ESX Server 3.5.X generell nicht unterstützt (VMware KB Article: 1008170).

Ein weitere Lösung für Paravirtualisierung ist UserMode Linux (UML) [Dik] im Separate-Kernel-Adress-Space-Modus. Auf weitere Details wird verzichtet, da dies nicht von Relevanz für die vorliegende Diplomarbeit ist.

Virtualisierung auf Betriebssystemebene

[opec] [Par]

Dieses Verfahren basiert auf der Partitionierung des vorhandenen Betriebssystems, jedoch nicht auf Virtualisierung der jeweiligen Hardware. Bei dieser Virtualisierungsart läuft ein modifizierter Host Kernel, welcher die virtuellen Einheiten (Virtual Environments) bereitstellt. Jede Virtual Environment (VE) läuft wie ein eigenständiger dedizierter Server, sicher und isoliert von anderen VEs auf dem physischen Hostsystem bei gemeinsamer Nutzung des Host Kernels. Aus Sicht des Hostbetriebssystems ist jede VE ein eigenständiger Prozess. Dabei wird für eine bessere Speicherausnutzung gesorgt und sichergestellt, dass laufende Applikationen nicht miteinander in Konflikt geraten. Im Vergleich zu anderen Virtualisierungsarten ist Betriebssystemvirtualisierung weniger flexibel, erzeugt hingegen aber weitaus weniger Overhead. Mittlerweile ist es sogar möglich verschiedene Linux-Distributionen innerhalb der VEs und unabhängig vom Hostbetriebssystem zu betreiben. Dafür bietet diese Technologie bessere Leistungsfähigkeit, Skalierbarkeit, dynamische Ressourcenverwaltung und einfachere Administration. Laut OpenVZ beträgt der Virtualisierungsaufwand nur 1-3% der Gesamtleistung des Systems.

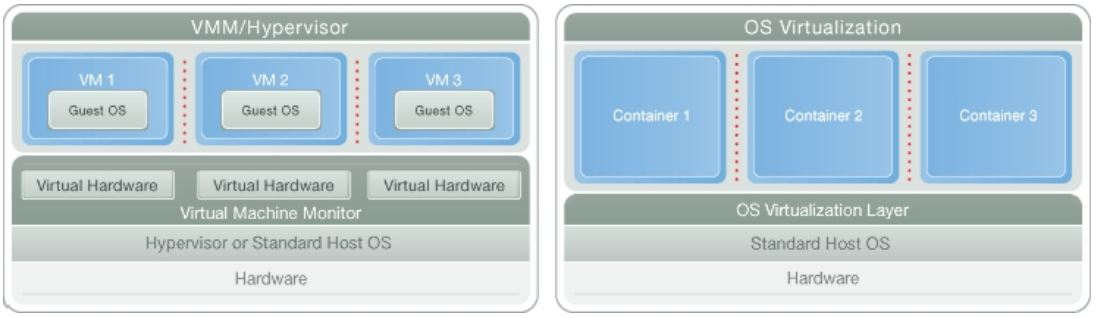

Abbildung 2.7: Vergleich HV-Virtualisierung / OS-Virtualisierung [Par]

Bekanntester Vertreter dieser Technologie ist Virtuozzo von Parallels (ehemals SWsoft), das erstmalig 2001 verfügbar war. Virtuozzo basiert auf der freien Variante OpenVZ (unterstützt durch Parallels), stellt jedoch ein komfortables Management Interface und viele weitere Features darüber hinaus zur Verfügung, z. B. den Einsatz von Microsoft Windows Server 2003/2008. Die Installation der Betriebssysteme in den VEs erfolgt über Templates (chroot-Container als tar-Archiv), so dass das Erstellen neuer VEs recht einfach zu handhaben ist. Auch die Wartung und Pflege ist aufgrund dieses Konzepts problemlos möglich. Dies betrifft vor allem das Sichern und Wiederherstellen einer VE. Die Ressourcenverwaltung spielt eine zentrale Rolle, da alle VEs ein und denselben Host-Kernel benutzen. Diese besteht im Wesentlichen aus Festplattenquota, CPU Scheduler sowie div. Ressourcenzähler (User Beancounters). Anders als bei anderen Virtualisierungslösungen

können all diese Parameter im laufenden Betrieb beeinflusst werden, so dass kein Neustart der jeweiligen VE erforderlich ist.

Jede VE, am Beispiel von OpenVZ, hat folgende eigenständige Objekte:

- **Dateien**
  Systembibliotheken, Applikationen, virtualisierte `/proc`, `/sys`, ...
- **Benutzer und Gruppen**
  Jedes VE hat den Benutzer "root" sowie auch andere Benutzer und Gruppen.
- **Prozessbaum**
  Ein VE kann nur eigene Prozesse sehen, die standardmäßig von init stammen. PIDs (Prozess-IDs) sind virtualisiert, so dass z. B. PID von init in jeder VE gleich 1 ist.
- **Netzwerk**
  Ein virtuelles Netzwerkgerät ermöglicht einer VE, eine eigene IP-Adresse zu besitzen, sowie ein Set von Netzwerkfiltern (iptables) und Routingtabellen.
- **Geräte**
  Falls nötig, kann für eine VE ein Zugang zu realen Geräten wie Netzwerk-Schnittstellen, seriellen Ports, Festplatten-Partitionen, usw. gewährleistet werden.
- **IPC-Objekte**
  Shared Memory, Semaphore, Messages

Weitere Vertreter der Betriebssystemvirtualisierung sind Linux-VServer, FreeBSD Jails, Solaris Containers und Linux Containers.

Besonders erwähnenswert ist Linux Containers, auch als LXC bezeichnet, welches erst vor kurzem die Aufnahme in den Linux-Kernel (ab Version 2.6.29) geschafft hat. Wie der Name bereits vermuten lässt, handelt es sich dabei auch um eine Art Container-Virtualisierung, die VEs bereitstellt. Ein interessantes HowTo und weiterführende Literatur dazu findet man unter [Hel09] sowie [McN09]. Wie sich das Projekt zukünftig entwickelt, bleibt abzuwarten - auf jeden Fall eine Alternative zum Linux-VServer Projekt, das die Aufnahme in den Linux-Kernel bis dato nicht geschafft hat.

Hardware-basierte Virtualisierung

[SBZD07] [Som]

Im Zuge zunehmender Serverkonsolidierung und damit verbundener Virtualisierung haben die Prozessorhersteller Intel und AMD Virtualisierungstechniken direkt auf Hardware-Basis in den Prozessor integriert. Beide Virtualisierungstechnologien (Intel VT-x, AMD-V) implementieren eine Secure Virtual Machine, basieren auf den gleichen Ansätzen, sind jedoch nicht kompatibel zueinander. In der ersten Ausbaustufe dieser Technologien ist es nun möglich, Gastbetriebssysteme ohne Anpassung in einer virtuellen Umgebung zu betreiben. Daher ist die hardware-basierte Virtualisierung auf der x86-Architektur eine Sonderform der Vollvirtualisierung, da hier keine Schnittstelle mehr geschaffen werden muss.

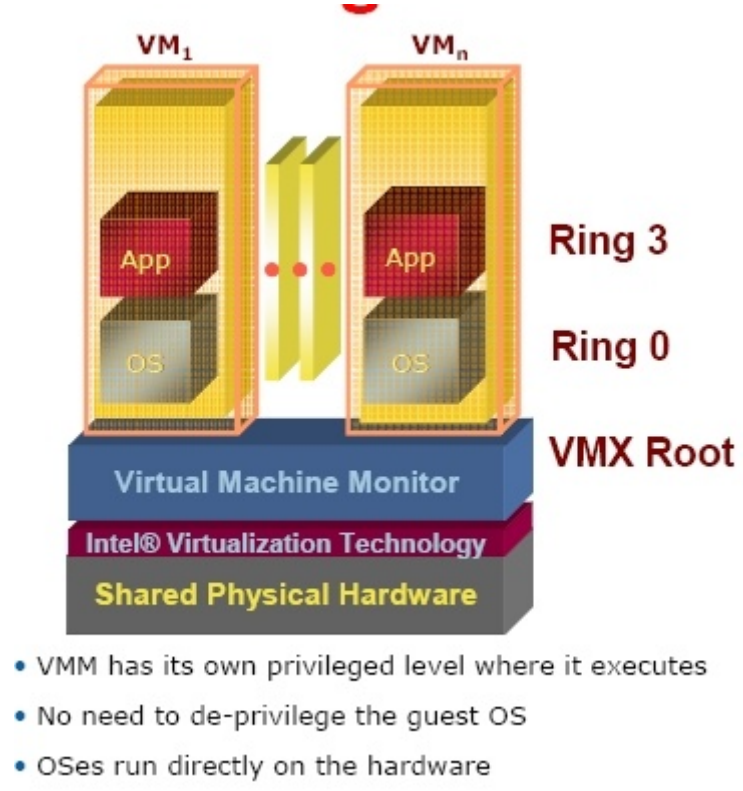

Abbildung 2.8: Intel Virtualization Technology VT-x [Som]

Auch XEN profitiert in der aktuellen Version von diesen hardware-basierenden Virtualisierungstechnologien, so dass nun auch ungepatchte Gastbetriebssysteme eingesetzt werden können - interessant vor allem für Produkte aus dem Hause von Microsoft. Da XEN jedoch von Citrix gekauft wurde und die OpenSource-Variante derzeit nur für Kernel 2.6.18 gepflegt wird, ist Kernel-based Virtual Machine (KVM) [KVM], das auf QEMU [QEM] aufsetzt und Hardwarevirtualisierung benötigt, eine ernstzunehmende Alternati-

ve. Einige Linux-Distributionen haben angekündigt XEN nur bis 2014 zu unterstützen und danach auf KVM zu setzen. Dies ist seit Version 2.6.20 fester Bestandteil des Linux-Kernels und liegt in Modulform vor, so dass kein Kernel-Patch und damit verbundener Neustart des Systems notwendig ist. Sobald das Modul geladen wurde, arbeitet der Linux-Kernel als HV. Auch die Unterstützung der Paravirtualisierung ist mittlerweile in KVM eingeflossen. Entwickelt wird KVM von Qumranet [RHE], einem israelischen Unternehmen, das im September 2008 von Red Hat gekauft wurde. Die zahlreich unterstützten Gastsysteme und weitere Details findet man auf der Projektseite unter [KVM].

In weiteren Ausbaustufen der Virtualisierungstechnologien kommt eine IOMMU zum Einsatz, welche direkt im Chipsatz integriert ist. Die Funktionsweise ist ähnlich der einer MMU für den Hauptspeicher. Sie dient zur Verbesserung der I/O Performance sämtlicher Geräte. Daher derzeit groß im Trend, aber noch in der Entwicklung steckt die Virtualisierung der Ein- und Ausgabegeräte. Das betrifft aktuell vor allem Netzwerkadapter und Massenspeicher. Bisher regelt der HV den Zugriff auf sämtliche I/O-Geräte. Sobald mehrere Gastsysteme gleichzeitig auf diese zugreifen möchten, kann nicht jedes auf beliebige Adressbereiche der jeweiligen Geräte schreiben, welche für die Kommunikation (Ports, DMA) mit dem Gerät genutzt werden. Dies würde in Kürze zu einem kompletten Chaos führen und die zu übermittelnden Informationen verfälschen. Wie die Abbildung 2.9 zeigt, sind die Ansätze der HV dafür unterschiedlich.

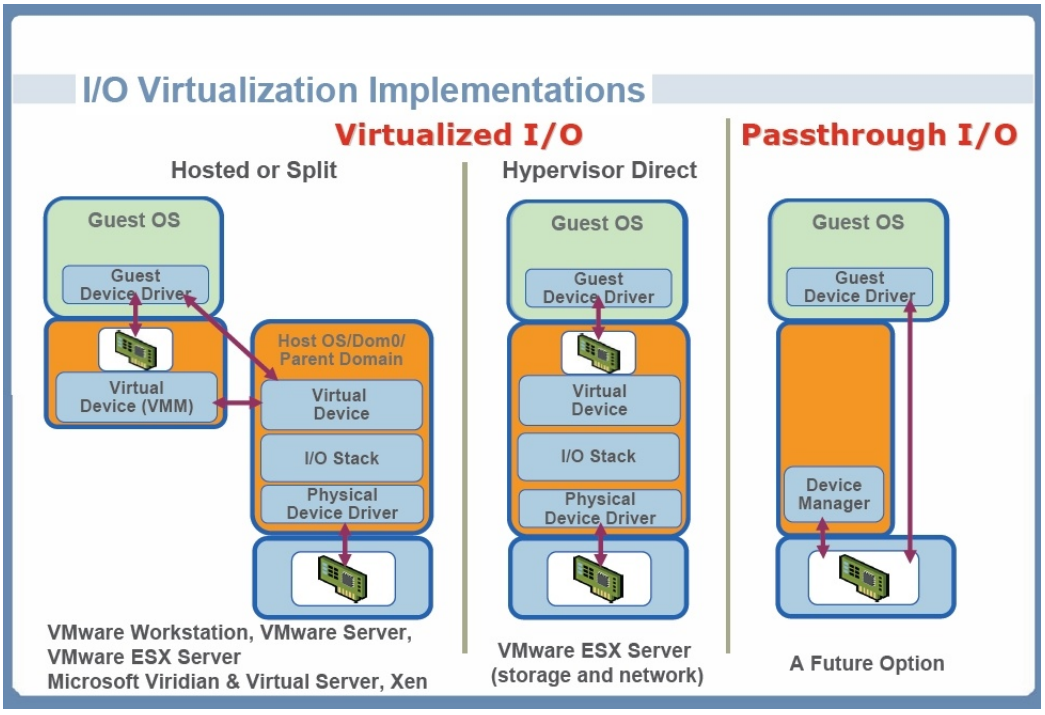

Abbildung 2.9: Möglichkeiten der I/O-Virtualisierung [Som]

VMWares ESX nutzt dafür Treiber, die speziell von den Hardwareherstellern für den HV entwickelt werden, durch die der Zugriff aus allen Gastsystemen erfolgt. Microsofts Hyper-V oder Citrixs XenServer hingegen leiten alle Zugriffe aus den VMs durch ein privilegiertes Service-Betriebssystem, das in einer eigenen Partition läuft. Durch den Einsatz von bevorzugt paravirtualisierten Treibern im Gastsystem kann der Datentransfer direkt und entsprechend schnell mit dem HV abgewickelt werden.

Mittlerweile haben sich auch die CPU-Hersteller Intel und AMD auf I/O-Virtualisierung spezialisiert, indem sie z. B. DMA-Puffer nach dem Muster von Nested Pages so einrichten, dass sie von allen VMs direkt angesprochen werden können. Intel bezeichnet diese weitere Ausbaustufe der Virtualisierungstechnologie als Intel VT-d (Virtualization for Directed I/O) [Int08] - beim Konkurrenten AMD dagegen heißt diese AMD-Vi (IOMMU) [AMD]. Mit Unterstützung dieser Technologien wird das Durchreichen eines PCI-Gerätes direkt an das Gastsystem durch eine I/O-Memory-Management-Unit (IOMMU) im Chipsatz möglich. Um z. B. die Netzwerkperformance zu verbessern, kann die Netzwerkkarte direkt an ein Gastsystem zugewiesen werden, was jedoch zu Einschränkungen bei der Migration von VMs führt, sofern die physische Ausstattung der einzelnen Serversysteme unterschiedlich ist.

Um letztendlich den HV zu entlasten und dadurch einen Performancegewinn zu erreichen, ist das korrekte Entgegennehmen, Sortieren und Verarbeiten von parallelen Anfragen an die jeweiligen Baugruppen erforderlich. Aufgrund der Hardwarevielfalt und zahlreicher Komponenten in der x86-Welt wird der weitere Ausbau der I/O-Virtualisierung noch einige Zeit in Anspruch nehmen.

Währenddessen behelfen sich Hersteller von Virtualisierungslösungen bei Performanceengpässen mit Direktzugriff auf I/O-Komponenten. VMwares DirectPath, eine neue Funktion in vSphere 4 [VG09], basiert auf dieser Idee und ermöglicht es einer VM direkt auf die physischen Hardwaregeräte zuzugreifen, um damit die CPU-Effizienz zu verbessern. Dafür ist die Nutzung von einigen Virtualisierungsfunktionen, wie z. B. VMotion, das die Hardwareunabhängigkeit und die gemeinsame Nutzung physischer E/A-Geräte erfordert, nicht möglich. Ferner ist diese Funktion im Moment auf ein paar wenige unterstützte Netzwerk- und Storage Adapter beschränkt, benötigt eine IOMMU (Intels VT-d bzw. AMDs-Vi) und eignet sich aufgrund der genannten Beschränkungen nicht zum Einsatz in hochverfügbaren Umgebungen.

2.2.3 Vorteile

In erster Linie erreicht man durch die Virtualisierung von Systemen eine Menge an Vorteilen - interessant ist dabei vor allem der Nutzen aus unterschiedlichen Betrachtungsweisen. Im Zeitalter von Green-IT [Han08] spielt die wirtschaftliche Nutzung von Ressourcen eine große Rolle. Mit der Einführung von Virtualisierungslösungen erreicht man eine optimale Serverauslastung, d. h., dass die zur Verfügung stehenden Ressourcen effizient genutzt werden können. Weiterhin ist die dynamische Zuteilung von Systemressourcen möglich, da kaum eine Anwendung dauerhaft die volle Rechenleistung über einen längeren Zeitraum benötigt, was sich wiederum positiv auf den Energieverbrauch und die damit verbundene Abwärme auswirkt. Dies spart im Weiteren aufwändige Kühllösungen. Diesen Prozess bezeichnet man auch als Serverkonsolidierung, bei dem die bestehende IT-Infrastruktur zusammengefasst wird, dabei die Anzahl der einzelnen physischen Maschinen reduziert wird und auf einen oder mehreren leistungsstarken Servern virtualisiert werden, um damit eine optimale Auslastung der vorhandenen Kapazität zu erreichen. Zum anderen senkt dieser Prozess die Kosten auf Dauer. Die Aufteilung der insgesamt zur Verfügung stehenden Ressourcen erfolgt je nach Virtualisierungssoftware automatisch oder kann bei Bedarf auch manuell zugewiesen werden.

Weiterhin verbessert sich die Wartbarkeit durch Virtualisierung sehr stark. Virtuelle Maschinen können einfach gesichert, problemlos auf einen anderen physischen Host ausgelagert werden und bei größeren Sicherheits- und Systemupdates jeglicher Art kann ein Snapshot vom Momentanzustand der Maschine erstellt werden, welcher im Problemfall wieder auf den Ausgangszustand zurückgesetzt werden kann. Auch der Umzug einer oder mehrerer VMs auf neue physische Hardware gelingt zumindest im Falle der Vollvirtualisierung problemlos, da hier eine definierte Hardwareumgebung mit Standardtreibern aus dem jeweiligen Betriebssystem repräsentiert wird. Das erspart dem Systemadministrator langwierige Anpassungen oder gar die Neuinstallation des Betriebssystems, was wiederum erhebliche Kosten und einen längeren Ausfall zur Folge hätte. Dies gilt natürlich auch für das Ersetzen von defekten Komponenten im jeweiligen Serversystem.

Der Grundgedanke, dass bestimmte Applikation sauber voneinander getrennt laufen sollen, lässt sich mit Virtualisierung perfekt umsetzen, ohne dabei unverhältnismäßig Ressourcen verschwenden zu müssen. Auch stellt der Betrieb unterschiedlichster Betriebssysteme bei Vollvirtualisierung kein Problem dar, so dass heterogene Umgebungen dadurch realisiert werden können, ohne dafür jeweils einen physischen Server betreiben zu müssen. Auch für Testumgebungen eignen sich VMs perfekt. Man kann diese schnell und problemlos, z. B. als Template, für den Anwender bereitstellen. Mit den bereits erwähnten Backup- und Snapshot-Möglichkeiten erreicht man auch hier einen Komfort für den

Anwender, der ohne Virtualisierung in dieser Form gar nicht oder nur sehr umständlich möglich wäre.

2.2.4 Nachteile

Bei all den genannten Vorteilen hat die Virtualisierung von Systemen selbstverständlich auch Nachteile. Anzumerken ist hierbei, dass die Virtualisierung kein Allheilmittel für sämtliche Probleme darstellt. Es bedarf einer guten Anforderungsplanung, was überhaupt in welchem Umfang virtualisiert werden soll. Gerade im Bereich des High-Performance-Computings hat Virtualisierung wenig Sinn, da oftmals die kompletten Ressourcen nahezu rund um die Uhr ausgeschöpft sind. Virtualisierung ist nur dann vorteilhaft, wenn viele Maschinen mit geringer oder mittlerer Systemauslastung auf einem physischen Rechner betrieben werden sollen. Hinzu kommt, dass die Virtualisierung selbstverständlich auch Systemressourcen benötigt. Ferner können nicht alle Hardwarekomponenten virtualisiert werden. Dies betrifft in der Regel Addon-Karten auf dem PCI-Bus und reicht bis hin zu Peripherieschnittstellen, wie z. B. USB beim ESX-Server, welche nicht an eine VM durchgereicht werden können. An dieser Stelle gibt es vor allem mit Lizenzservern Probleme, die verschiedenste Hardware-Dongles als Basis nutzen.

Ein weitere größere Thematik stellt die Sicherheit in virtuellen Umgebungen dar. Da diesem Punkt besonderes Augenmerk geschenkt werden soll, werden Details dazu im nächsten Kapitel 2.2.5 näher betrachtet. Bedingt dadurch wird der HV zum SPOF, was als sehr kritisch betrachtet werden sollte.

Bei kommerziellen Lösungen können hohe Lizenzkosten entstehen, welche in der Projektkalkulation entsprechend zu berücksichtigen sind. Die Grundfunktionalität zum Betrieb eines einzelnen Virtualisierungsservers wird meist kostenfrei von allen Anbietern zur Verfügung gestellt. Sobald aber eine Farm aus Servern betrieben werden soll, um damit die ganzen Vorzüge der Virtualisierung nutzen zu können, werden Lizenzkosten fällig. Das betrifft im Allgemeinen Zusatzfunktionen, wie z. B. das Verschieben von virtuellen Maschinen über physische Hosts, dynamischer Lastausgleich, Hochverfügbarkeit, Virtualisierungsmanagement und Backup. Je nach gewähltem Lizenztyp erhält man eine Komplettlösung für verschiedene Problemkategorien auf Basis der Virtualisierung. An dieser Stelle muss natürlich differenziert werden, inwieweit die Virtualisierungslösung im jeweiligen Einsatzgebiet sinnvoll mit dem gebotenen Funktionsumfang eingesetzt werden kann und ob das die hohen Lizenzkosten rechtfertigt.

2.2.5 Sicherheit

Oftmals wird die Sicherheit in virtuellen Umgebungen komplett vernachlässigt bzw. nicht bedacht. Das regelmäßige rechtzeitige Einspielen von Patches gehört zu den Pflichtaufgaben eines jeden Systemadministrators. Dennoch können bisher unentdeckte Sicherheitslücken, vor allem im HV, zur Kompromittierung des Gesamtsystems führen. Besonders kritisch ist dabei das Ausbrechen aus den virtuellen Maschinen. Wie in [Gru09] beschrieben, war es vor kurzem unter VMwares ESX und VMwares Workstation möglich, unbegrenzt Speicher vom VMware-Host in den Gast zu kopieren, dort Daten zu lesen sowie vom Gastsystem auf Speicher vom Host zugreifen, welches noch ein viel kritischeres Problem darstellt. Darüber konnte sich ein Angreifer einer VM direkten Zugriff auf den Host verschaffen. Dieses Problem ist mittlerweile unter dem Namen SVGA_CMD_RECT_COPY-Bug bekannt und wurde seitens VMware stillschweigend durch einen Patch behoben. Dies hinterlässt einen faden Beigeschmack, da dies durchweg alle VMware-Produkte betraf und heimlich beseitigt wurde.

Trotz der Unterstützung der Virtualisierung auf Hardwareebene, direkt in der CPU durch die Prozessorhersteller, benötigt eine VM I/O-Geräte, Videoadapter, Speicher-Controller, Netz- und COM- / PAR-Adapter, welche die Virtualisierungslösung per Software bereitstellen muss. Diese werden größtenteils über Speicherbereiche im Host emuliert, der wiederum als virtueller Speicher in das Gastsystem eingeblendet wird. VMwares vSphere 4, Novells XenServer sowie Microsofts Hyper-V basieren alle auf diesem Prinzip. Sobald es dort oder in einem Treiber des Gastsystems zu einer Fehlfunktion kommt, ist es möglich, aus dem Gastsystem auszubrechen und in den Speicherbereich des Hosts zu schreiben. Dies war im Falle von VMware geschehen, als durch einen Softwarefehler in der 3D-Beschleunigung des virtuellen VGA-Adapters direkter Zugriff auf den Hostspeicher möglich wurde.

Es sollte bedacht werden, dass die Emulation von virtuellen Geräten jeglicher Art unvorhersehbare Sicherheitsrisiken bergen, die oftmals unterschätzt werden. Dies führt letztendlich zu weniger Sicherheit in virtuellen Umgebungen und verstärkt die Angriffsgefahr. Deshalb ist ein ausgereiftes Sicherheitskonzept nötig - es ist zwingend davon abzuraten, auf einem physischen Host VMs mit unterschiedlichen Sicherheitsstufen zu betreiben, auch wenn die Hersteller das Gegenteil behaupten, dass die Software bzw. die Verwaltungsschicht (HV) ausreichend sicher sei und unterschiedliche Sicherheitsstufen innerhalb der gleichen Maschine daher kein Problem darstellen. Es sollte jedem klar sein, dass auch hier immer wieder neue Sicherheitslücken auftreten werden und man den Herstellerangaben nur wenig Glauben schenken sollte.

2.3 Hochverfügbarkeit versus Virtualisierung

Prinzipiell schließen sich Hochverfügbarkeit und Virtualisierung gegenseitig aus, da die Virtualisierung von Systemen die Verfügbarkeit senkt [Len09]. Ein defekter physischer Server würde dabei eine ganze Gruppe von virtuellen Umgebungen mit sich ziehen, die dann durch mehr oder wenig technischen Aufwand wieder in Betrieb genommen werden müssten. Erst bei Aufteilung der kritischen VMs auf verschiedene physische Hosts kann die Verfügbarkeit dadurch wieder kompensiert werden. Dabei spielt die geschickte Verteilung der angebotenen Dienste über die komplette Serverfarm eine große Rolle, so dass auch bei Ausfall eines einzelnen Servers der Anwender keine oder nur eine kurze Beeinträchtigung wahrnimmt. Sichert man die virtuellen Maschinen auf verschiedene Ebenen ab, so lassen sich dadurch wieder hochverfügbare Dienste realisieren. Das bedeutet im Detail, dass man zum einen die High Availability Funktionen der Virtualisierungslösung nutzt, welche im Fehlerfall des gesamten physischen Hosts oder der jeweiligen VM die Betroffene(n) auf einem anderen Server im Ressourcenpool neu startet und zum anderen auf Applikationsebene den jeweils kritischen Dienst absichert und redundant über weitere virtuelle Maschinen mit einer Clusterlösung (z. B. Linux-HA) absichert, so dass ein einwandfreier Betrieb gewährleistet werden kann.

Verfügt man aus lizenztechnischen Gründen nicht über die Hochverfügbarkeitsfunktionen der Virtualisierungssoftware, müssen die gesamten Ressourcen auf Applikationsebene abgesichert werden. Dazu eignet sich die kostenfreie Cluster Suite Linux-HA ab Version 2 [Linb], welche bis zu 16 Knoten in einem Cluster unterstützt. Da die Konfiguration solcher Cluster nicht trivial ist, empfiehlt es sich, die HA-Funktionen der Virtualisierungslösung zu nutzen, so dass man in der Regel mit einem Failover-Cluster auf Applikationsebene auskommt, welcher auch schon in Linux-HA Version 1 unterstützt wird. Zusätzlich dazu ist es zwingend notwendig in beiden Fällen Monitoring-Software einzusetzen, die den Serverbetreiber im Fehlerfall per E-Mail oder SMS benachrichtigt, um das korrekte Arbeiten der bereitgestellten Dienste sicherzustellen und um ggf. entsprechende Maßnahmen zur Vermeidung erneuter Ausfälle gleicher Art einleiten zu können. An dieser Stelle muss noch einmal ausdrücklich darauf hingewiesen werden, dass die HA-Features der Virtualisierungssoftware oftmals kein Monitoring einzelner Dienste vornehmen, sondern nur den physischen Host bzw. die einzelne VM als Gesamteinheit überwachen.

Aus diesem Grund ist eine externe Überwachung (siehe auch Kapitel 3.1.3) auf Applikationsebene ratsam, damit auch im Fehlerfall reagiert werden kann. Wie hier unschwer zu erkennen ist, kann Virtualisierung und Hochverfügbarkeit erst mit einer Serverfarm erreicht werden. Es sollte dennoch bedacht werden, dass sich die Zuverlässigkeit des

Gesamtsystems bereits durch die Konsolidierung alter Desktop Hardware auf aktuelle Serverhardware erhöht. Moderne Serversysteme gelten als fehlertolerant, da diese über redundante Hardwarekomponenten verfügen. In welchem Umfang die Redundanz der jeweiligen Komponenten erforderlich ist, hängt wieder vom Einsatzgebiet, dem zur Verfügung stehenden Kapital und welche Verfügbarkeitsklasse über das Jahresmittel damit erreicht werden soll, ab.

2.4 Clustertheorie

[Bre07] [Sch08] [Sch09]

2.4.1 Allgemeines und Definition

Der Zusammenschluss von Computersystemen über ein Netzwerk zu einem bestimmten Zweck wird als Cluster bezeichnet. Die gemeinsam zu bewältigende Aufgabe kann dabei sehr unterschiedlich sein und klassifiziert im Allgemeinen die Art des Clusters. Von extern betrachtet, kann ein Cluster in der Regel als Einzelsystem angesehen werden. Ein Cluster dient meistens zur Erhöhung der Rechenleistung oder verbessert die Verfügbarkeit im Vergleich zu einem einzelnen Computersystem. Jeder einzelne Rechner wird als Knoten (engl. Node) in einem Cluster bezeichnet. Die häufigsten Cluster bestehen aus einer Anzahl von n Knoten sowie einem Cluster Manager (CM), welcher die Ressourcen verwaltet und die Arbeit koordiniert. Die erforderlichen Informationen werden gewöhnlich von jedem Knoten vorgehalten. Der Austausch dieser erfolgt über einen clusterinternen Message Stack.

Dennoch geht der Trend vom traditionellen Clustern, ausgenommen im High-Performance-Computing, etwas zurück, da der Kostendruck immer mehr steigt und Sparmaßnahmen erforderlich sind. Man gibt sich allgemein mit weniger Verfügbarkeit zufrieden und schätzt das Risiko eines Ausfalles genau ab. Hier kann man optimal mit der Virtualisierung von Systemen ansetzen, indem man einen kritischen Dienst (z. B. Load Balancer) auf zwei verschiedene physische Maschinen aufteilt und als Failover-Cluster auf Applikationsebene absichert. Dadurch wird zusätzlich die Komplexität des Clustersystems auf überschaubarem Niveau gehalten.

2.4.2 Grundtypen

Load Balancing-Cluster

Load Balancing-Cluster (SLB) werden zur Verteilung der Last auf verschiedene Systeme eingesetzt, so dass jeder Knoten einen Teil einer Aufgabe übernimmt. Diese Clusterart trifft man vor allem im Webhosting-Bereich und Datenbankumfeld an. Ein Cluster Manager (CM) verteilt die eingehenden Anfragen auf sämtliche Knoten, auf denen die Applikationen laufen. Diese arbeiten ihren Teil jeweils autonom ab. Die redundante Auslegung des CM ist zwingend erforderlich, da hier ein SPOF existiert. Die dynamische Lastverteilung erfolgt nach festgelegten Regeln sowie div. Scheduling-Algorithmen, die im CM konfiguriert werden können. Der Ausfall eines Applikationsknotens führt nicht zum Gesamtausfall des Systems. Je nach Dimensionierung des Gesamtclusters können sich dadurch Performanceeinbußen ergeben. Der Leistungsbedarf kann jedoch einfach mit Hinzufügen von einzelnen Knoten beliebig erweitert werden. Daher ist diese Clusterart in Umgebungen mit sich ständig änderndem Lastverhalten sehr beliebt.

HA-Cluster

Hochverfügbarkeitscluster (HA-Cluster) werden zur Verbesserung der Verfügbarkeit eingesetzt, um damit die Wahrscheinlichkeit für einen Ausfall zu verringern. Da sich diese Diplomarbeit vorrangig mit HA-Clustern beschäftigt, wird jene Clusterart nachfolgend näher ausgeführt. Ein HA-Cluster dient einzig und allein der Hochverfügbarkeit, jedoch nicht der Lastverteilung oder gar höherer Performance. Auch beide anderen vorgestellten Clusterarten verkraften den Ausfall einzelner Knoten ohne größere Probleme - der Fokus beim HA-Cluster liegt hingegen beim Ausfall eines Knoten. Die Mitglieder eines HA-Clusters überwachen sich gegenseitig bzw. werden extern überwacht. Bei Ausfall eines Dienstes oder eines kompletten Knotens übernimmt ein anderer dessen Aufgaben bzw. Teile davon. Die meisten HA-Cluster bestehen aus zwei Knoten, auch als Failover- bzw. Active/Passive-Cluster bekannt.

Innerhalb eines HA-Clusters müssen alle Knoten auf einen gemeinsamen Datenbestand zugreifen können oder die benötigten Daten in irgendeiner Form replizieren. Bei HA-Clustern mit umfangreichen Datenbestand, ist die Replikation dieser oft nicht mehr zweckmäßig realisierbar, so dass der Einsatz eines Shared Storage in Form eines Speichernetzwerkes (SAN, NAS oder iSCSI-Host) unumgänglich ist.

Im Bereich der HA-Cluster unterschiedet man die vier folgenden Subtypen:

Active/Passive-Cluster (HA)

Ein Active/Passive- (siehe Abbildung 2.10), zum Teil auch als Failover-, Master/Slave-Cluster oder Hot Standby bezeichnet, besteht in der Regel aus zwei Knoten und stellt die einfachste Clusterform dar.

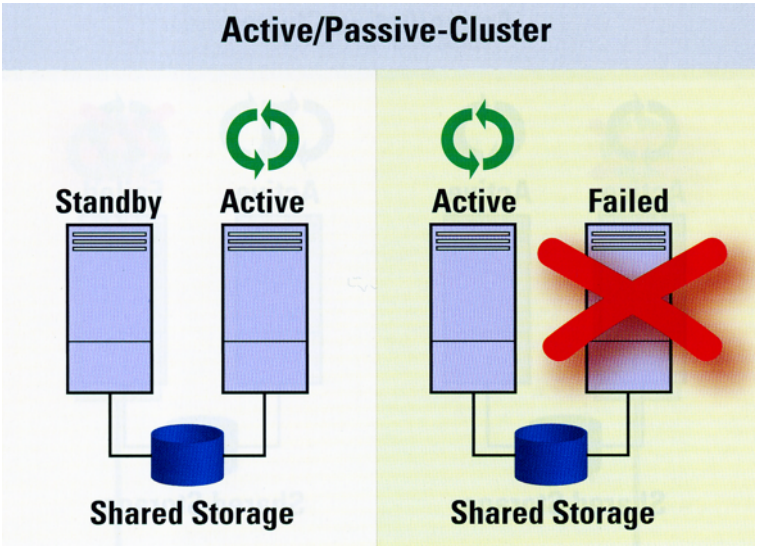

Abbildung 2.10: Active/Passive-Cluster [Bre07]

Dabei überwacht der Slave den Master über eine spezielle Netzwerkverbindung (auch serielle Verbindung über die COM-Schnittstelle möglich) und sendet diesem kleine Netzwerkpakete, den sogenannten Heartbeat, die die Gegenseite beantwortet. Bleibt die Antwort aus, erklärt der Slave seinen Master nach einer konfigurierbaren Zeit für tot und übernimmt dessen Ressourcen (IP-Adresse und konfigurierten Dienste), die der Master zuvor angeboten hat. Dieser Vorgang wird als Failover bezeichnet, der dieser Clusterart auch ihren Namen gibt. Das Prinzip ist denkbar einfach und unkompliziert, hat aber den Nachteil, dass die Ressourcen des Slaves brach liegen, solange dieser nur Beobachter ist.

Dieser Nachteil lässt sich aber mithilfe der Virtualisierung ausgleichen, indem beide Knoten auf unterschiedliche Hosts verteilt werden, auf denen noch andere VMs im Betrieb sind und die Ressourcen damit dennoch effizient genutzt werden können. Die dynamische Ressourcenverteilung der Virtualisierung ist dafür optimal geeignet.

Active/Active-Cluster (HA)

Active/Active-Cluster (siehe Abbildung 2.11) nutzen die Ressourcen beider Knoten besser aus, indem jeder Knoten unterschiedliche Dienste anbietet. Die Knoten überwachen ihren Allgemeinzustand nicht gegenseitig, sondern prüfen gegenseitig die angebotenen Dienste. Ist einer davon nicht mehr verfügbar, übernimmt jeweils der andere Knoten diesen. Anzumerken ist hierbei noch, dass im Normalfall zu keiner Zeit ein Dienst gleichzeitig von beiden Knoten angeboten wird.

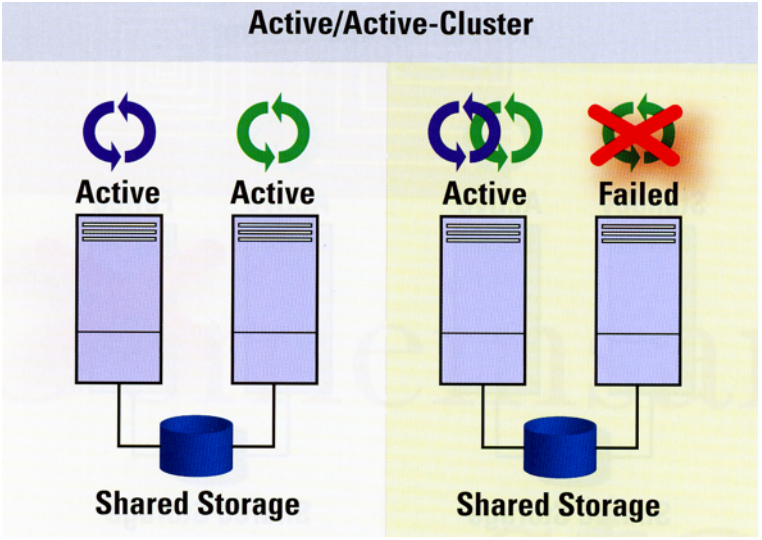

Abbildung 2.11: Active/Active-Cluster [Bre07]

Weil ein Service im Regelfall nicht nur aus einer Ressource besteht, werden diese im Cluster zu einer Gruppe zusammengefasst. Dazu zählt z. B. die IP-Adresse unter der ein Dienst erreichbar ist oder ein Dateisystem, auf das zugegriffen werden muss. Im Fehlerfall verlagert der Cluster die komplette betroffene Gruppe. Daran kann man auch ein Regelwerk von Bedingungen knüpfen, z. B. in welcher Reihenfolge die Dienste auf dem anderen Knoten gestartet werden sollen. Weiterhin lässt sich damit eine Hierarchie von Abhängigkeiten realisieren, so dass weitere Ressourcen verlagert werden können, die nicht unmittelbar mit dem fehlerhaften Dienst im Zusammenhang stehen.

M-to-N-Cluster (HA)

Dieser Clustertyp (siehe Abbildung 2.12) stellt eine Erweiterung eines Active/Active-Clusters mit zwei Knoten dar. Hier verteilen sich eine Anzahl m Ressourcen auf eine Anzahl von n Knoten. Ein M-to-N-Cluster ist die komplexeste Clusterart und umfasst in der Praxis nicht mehr als 16 Knoten, da dieser sonst nicht mehr beherrschbar wäre.

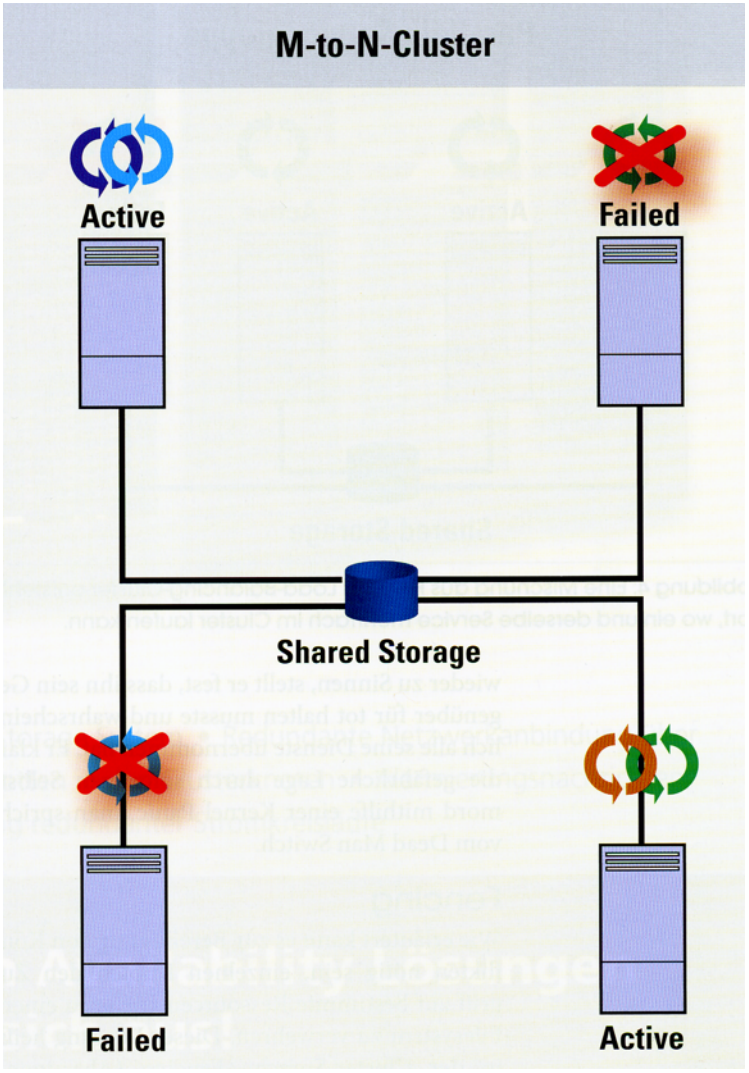

Abbildung 2.12: M-to-N-Cluster [Bre07]

Wer im Fehlerfall für wen einspringt und auf welchem Knoten die Ressourcen verlagert werden können, bestimmen Richtlinien, die unter Umständen sehr kompliziert sein können. Bei den meisten Clusterlösungen wird ein Punktesystem dafür benutzt, bei dem ein Dienst auf einen Knoten umzieht, auf dem er die meisten Punkte erhalten würde. Andere Clusterlösungen hingegen knüpfen die Übernahme an Bedingungen, wie z. B. einen Maximalwert der Auslastung auf dem Zielsystem oder bereits laufende erwünschte oder unerwünschte Nachbarapplikationen auf dem Node, der einen Dienst übernehmen soll.

Der Vorteil eines M-to-N-Clusters liegt in der besseren Hardwareauslastung und höherer Ausfallsicherheit, was für den Einsatz in virtuellen Umgebungen nicht weiter von Bedeutung ist. Aufgrund der Komplexität ist davon abzuraten in virtualisierten Umgebungen eine solche Clusterart umzusetzen, da dies zum einen aufgrund der Möglichkeiten der Virtualisierung nicht erforderlich ist und zum anderen nicht mehr überschaubar und sinnvoll wartbar ist, sobald die Live-Migration der Virtualisierungslösung mit dem Umzug von Ressourcen der Cluster Suite assoziiert wird. Es ist dennoch denkbar, falls der Cluster Manager Kenntnis davon hat, dass es sich dabei um virtuelle Maschinen handelt. Mittlerweile existieren in der Clusterlösung Linux HA V2/V3 OCF-Agenten, die eine Kombination mit Xen, OpenVZ oder VMware Server ermöglichen würden.

Ein Nachteil dieser Clusterart besteht in der Datenhaltung. Da mehr als zwei Knoten an diesem Cluster beteiligt sind, ist eine Replikation der Daten nicht mehr sinnvoll möglich. Ein externes Storage-System wird erforderlich, damit alle Nodes auf einen gemeinsamen Datenbereich zugreifen können.

Parallel Service Groups (HA+SLB)

Eine parallele Servicegruppe (siehe Abbildung 2.13) ist eine Mischung aus HA- und Load Balancing-Cluster. Hier läuft derselbe Dienst auf mehreren Knoten, über dem sich eingehende Anfragen verteilen.

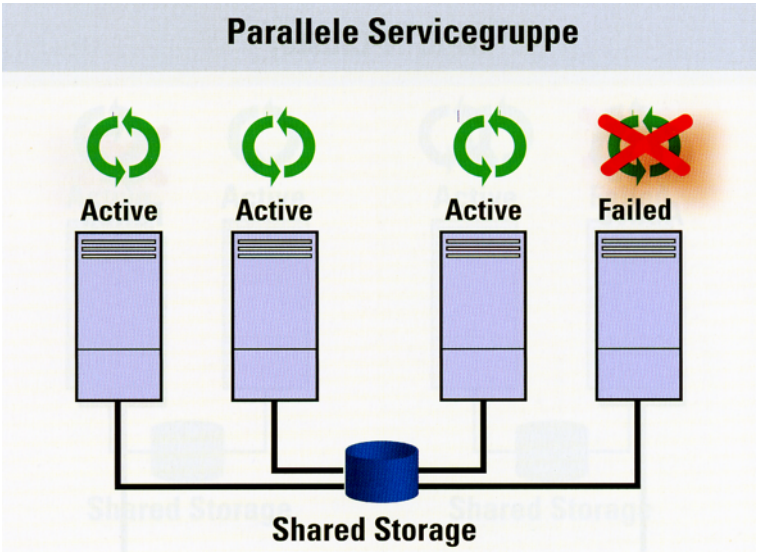

Abbildung 2.13: Parallele Servicegruppe [Bre07]

Man erreicht dadurch eine hohe Verfügbarkeit bei guter Lastverteilung. Nachteilig ist auch hier wieder die Notwendigkeit eines externen Storage-Systems in Verbindung mit einem Clusterdateisystem, welches die parallelen Zugriffe mithilfe eines Lock Managers organisiert.

HPC-Cluster

High-Performance-Computing-Cluster dienen zur Berechnung komplexer Probleme, die mit einem Einzelsystem in endlicher Zeit nahezu undenkbar wären. Heutzutage werden diese als Alternative zu monolithischen Mainframes (Großrechner) für rechen- und datenintensive Applikationen eingesetzt. Die schnellsten HPC-Cluster der Welt werden in einer Top 500 Liste [top] geführt und sind bei weitem keine Einzelstücke mehr, sondern bestehen aus einem ganzen Rechnerverbund. Als Betriebssystem nutzen die meisten HPC-Cluster Linux bzw. Derivate davon.

Die Rechenaufgaben werden über die Knoten verteilt. Die Verteilung der Aufgaben übernimmt gewöhnlich ein Job Managementsystem, auch als Batch- oder Queueing-System bezeichnet. Ein bekannter Vertreter in diesem Bereich ist die Sun Grid Engine. Die schnelle clusterinterne Kommunikation geschieht im Allgemeinen über das Message Passing Interface (MPI). Auch davon existieren eine Reihe von freien Implementierungen, wie z. B. OpenMPI. Die Kopplung einzelner Nodes erfolgt über ein serielles Hochgeschwindigkeits-Interconnect-Netzwerk (häufig InfiniBand), da ein schneller Datenaustausch zwischen den Knoten erforderlich ist. Die Anforderungen dafür sind geringe Latenzzeiten im Nanosekundenbereich und eine hohe Bandbreite, die bei InfiniBand QDR mittlerweile bis zu 40 GBit/s beträgt. An dieser Stelle erkennt man klar die Vorteile gegenüber dem gebräuchlichen TCP/IP-Ethernet.

Damit die Skalierbarkeit der rechenintensiven Aufgaben über das gesamte Clustersystem gewährleistet ist, erfolgt die Aufteilung des Problems in Teilprobleme. Die Ergebnisse werden am Ende wieder zusammengeführt. Viele kleine Optimierungen durch den Programmierer können die Skalierbarkeit jedoch noch erheblich verbessern.

Am meisten werden HPC-Cluster zur Realisierung unterschiedlichster Rechenaufgaben in Wissenschaft, Forschung und Industrie eingesetzt. Dazu zählen Strömungsberechnungen, Wetter- und Klimaprognose, Genanalysen, Materialforschung, Nachweis von Theorien der Atomphysik bis hin zu Crash-Simulationen. Der Aufgabenbereich ist vielfältig und für die Lösung eines solchen Problems würde ein einzelner Computer zum Teil Jahrzehnte benötigen.

2.4.3 Begriffsklärung

Im Zusammenhang mit Clustersystemen tauchen immer wieder einige Begriffe auf, die bei der Inbetriebnahme und Konfiguration von Clustersystemen unerlässlich sind und abschließend erklärt werden sollen.

Split Brain

Als Split Brain bezeichnet man die Störung der Kommunikation zwischen den einzelnen Knoten innerhalb eines Clustersystems. In Folge dessen zerfällt ein Mehr-Knoten-Cluster in zwei gleich oder ungleich große Teilcluster. Weiterhin besteht die Möglichkeit, dass mehrere Teilcluster entstehen. Die Betrachtung erfolgt dann jedoch über mehrere einzelne Split Brain-Szenarien.

Jeder Teil des Clusters nimmt an, dass der verbliebene Rest nicht mehr verfügbar ist und startet die Ressourcen, für die der jeweils andere Teilcluster zuständig ist. Im unkritischsten Fall führt dies dazu, dass mehrere Knoten mit der gleichen IP-Adresse im Netzwerk fungieren. Im Worst Case kann dies zu einer Zerstörung des kompletten Datenbestandes führen, weil in diesem Zustand die Datenintegrität nicht mehr gewährleistet werden kann.

Mithilfe eines Quorums wird versucht, solche Situationen zu verhindern. Weiterhin existieren zusätzlich noch alternative Methoden, wie z. B. Fencing, zur Vermeidung dieser. Ferner ist es ratsam, das Netzwerk für die clusterinterne Kommunikation redundant auszulegen, um einer Split Brain-Situation vorzubeugen. Weiterhin empfiehlt sich der gleichzeitige Einsatz eines Quorums und Cluster Interconnects. Erst dadurch wird eine Erkennung zwischen Teilung und partiellem Ausfall möglich.

Quorum

Zur Abwendung von Split Brain-Situationen kann ein Quorum in einem Clustersystem konfiguriert werden. Es existieren dabei verschiedene Prinzipien, die je nach eingesetzter Clusterlösung zur Verfügung stehen: [Mica]

- Node Majority
- Node and Disk Majority
- Node and File Share Majority
- No Majority: Disk Only

Beim Prinzip der Mehrheitsfindung (Node Majority), auch von der Cluster Suite Linux-HA genutzt, hält jeder Knoten lokal eine Information, die zur Entscheidungsfindung hilft.

Jeder Node darf hierbei seine Stimme (Voting) abgeben. In einem Cluster aus einer Anzahl von N Knoten wird der Teilcluster mit n Knoten primär, wenn gilt:

$$n > \lfloor \frac{N}{2} \rfloor$$

Das bedeutet, sobald mehr als die Hälfte aller Knoten des Gesamtclusters Mitglieder des Teilclusters sind. Der primäre Teilcluster übernimmt dann alle Ressourcen und führt die Aufgabe des Gesamtclusters fort. Die Nodes im Cluster, die kein Quorum haben, fahren alle Ressourcen solange herunter, bis die Kommunikation zum Rest wiederhergestellt worden ist. Das genaue Verhalten ist jedoch individuell konfigurierbar. In Linux-HA übernimmt das Open Cluster Framework [Spe] die Theorie zu Teilclustern. Anstelle der reinen Mehrheitsentscheidung ist weiterhin ein Pluggable Quorum Framework implementiert, das die Entscheidung durch zusätzliche Informationen treffen kann, welche hier nicht näher beschrieben werden sollen.

Außerdem kann die Information zur Mehrheitsfindung der jeweiligen Knoten auf einem Shared Storage, auch als Voting Disk bezeichnet, abgelegt werden. Alternativ kann dafür auch eine Dateifreigabe genutzt werden. Da das Shared Storage bzw. die Dateifreigabe wieder einen SPOF darstellt, ist diese Methode nicht empfehlenswert. Das am meisten eingesetzte Verfahren ist Node and Disk Majority, eine Kombination aus beiden erläuterten Prinzipien.

Fencing

Als Fencing (Abzäunung) bezeichnet man den Ausschluss von Ressourcen, deren Zustand unbestimmt ist, sobald der Cluster keine Informationen über seine Partnerknoten erhält. Dabei wird angenommen, dass der betroffene Knoten nicht mehr korrekt funktioniert. Um dies zu erreichen, existieren eine Reihe von Techniken. Diese reichen vom Ausschluss einzelner Ressourcen, wie z. B. dem Dateisystem, so dass ein Knoten zur Wahrung der Datenintegrität nicht mehr auf das Shared Storage zugreifen darf, bis hin zur kompletten Abschaltung des Knotens. Letzteres ist die einfachere Variante und wird auch als STONITH (Shoot The Other Node In The Head) bezeichnet, welche auch in Linux-HA genutzt wird und durch den noch funktionierenden primären Teilcluster erfolgt. Dazu existieren eine Reihe von STONITH-Agenten, um z. B. die Stromversorgung des Knotens über die unterbrechungsfreie Stromversorgung (USV) abzustellen oder per Intelligent Platform Management Interface (IPMI) einfach die Abschaltung des Knotens zu veranlassen.

Kapitel 3

Hochverfügbarkeit mit Linux

3.1 Allgemeine Anforderungen

3.1.1 Lokale Ausfallsicherheit

Bei der Konzeption eines Serversystems ist eine solide Hardwareauswahl wichtig. In hochverfügbaren Systemen sollte dabei auf eine redundante Stromversorgung, Arbeitsspeicher [Hal08] (ECC, Chipkill, Memory Mirroring) und Storage Controller besonderer Wert gelegt werden. Die Redundanz weiterer Hardwarekomponenten ist vom Einsatz und von der Relevanz des Systems abhängig. Für das Storage (lokal oder extern) ist die Verwendung eines RAID-Levels erforderlich. Gewöhnlich wird RAID 1 (Mirroring) bei zwei Festplatten oder RAID 5 bzw. RAID 6 bei der Verteilung von Daten auf mehreren Festplatten im Normalfall in einem Shared Storage eingesetzt. RAID 5 erlaubt den Ausfall einer Festplatte, RAID 6 sogar den von zwei. [LVH09]

Eine automatisierte Systeminstallation ist von Vorteil, damit ein Server schnell wieder funktionsfähig wird. Nicht immer werden Fehler von Hardwarekomponenten verursacht - oft liegen diese an den Wechselwirkungen zwischen Softwarekomponenten nach Installationen oder Updates. Im Linux-Bereich existieren dafür verschiedene Mechanismen um ein System ohne manuelles Eingreifen zu installieren, die oft distributionsabhängig sind. Auch die Wahl eines geeigneten Dateisystems während des Installationsprozesses ist für die lokale Ausfallsicherheit relevant.

3.1.2 Netzwerkbasierte Ausfallsicherheit

Im Weiteren ist die netzwerkbasierte Ausfallsicherheit zu gewährleisten, da oft die Redundanz der Netzwerkkomponenten vergessen wird. In vielen Serversystemen ist die Netzwerkkarte ein SPOF. Abhilfe schafft hier Ethernet Bonding, auch als Trunking, Link Aggregation oder Channeling bezeichnet, bei dem mehrere physische Netzwerkkarten zu einem virtuellen Netzwerkadapter zusammengefasst werden können. Der Linux-Kernel liefert die dafür nötigen Werkzeuge bereits mit. Bei Ausfall einer Netzwerkkarte übernimmt der verbleibende Rest einfach den Dienst. Das hat zum einen den Vorteil, dass keine manuelle Umkonfiguration erforderlich ist und zum anderen auch eine höhere Bandbreite verfügbar ist, solange keine der Netzwerkkarten im Verbund ausfällt. Diese Funktionalität hängt aber letztendlich von der Konfiguration des Bonding-Treibers ab.

Auf allen weiteren Netzwerkebenen kann eine Redundanz der jeweiligen Komponenten erfolgen. Bei Verwendung von redundant ausgelegten Switches sorgt das Spanning Tree Protocol (STP) für die Vermeidung von redundanten Netzwerkpfaden (Schleifen), indem es einen Spannbaum aufbaut. Mittlerweile existieren moderne Formen des STP, wie z. B. das Rapid Spanning Tree Protocol (RSTP) oder Multiple Spanning Tree Protocol (MSTP). Aufgrund der verbesserten Funktionsweise können dadurch die Ausfallzeiten eines Netzwerks erheblich reduziert werden oder voneinander unabhängig STP-Instanzen für ein VLAN oder eine Gruppe von VLANs beim MSTP gebildet werden.

Zum Einsatz redundanter Router fasst das Virtual Router Redundancy Protocol (VRRP) mehrere zu einer logischen Gruppe zusammen, die sich dem Netzwerk als ein Router repräsentiert. Dadurch kann eine schnelle Fehlerbehebung und automatisiertes Recovery ohne Rekonfiguration der jeweiligen Router im Fehlerfall stattfinden. Die Funktionalität des VRRP ähnelt Ciscos Hot Standby Router Protocol (HSRP), ist jedoch nicht proprietär.

Nach Möglichkeit sollte die clusterinterne Kommunikation vom restlichen Netz getrennt sein, da im Falle einer starken Auslastung diese gestört werden kann. Eine serielle Verbindung auf Basis der COM-Schnittstelle sollte nicht mehr verwendet werden, da diese nicht mehr zeitgemäß ist und auch von den neueren Kommunikationsstacks nicht mehr unterstützt wird. In der Praxis wird oft Ethernet Bonding für die Knoten genutzt, um dennoch ein gemeinsames Netzwerk nutzen zu können, aber andererseits auch genug Performance sicherzustellen. Letztendlich ist die Wahl der geeigneten Verbindung von der Zielsetzung abhängig.

3.1.3 Monitoring

[Sch09]

Neben dem eigentlichen Betrieb des Clustersystems ist die externe Überwachung der Dienste in hochverfügbaren Systemen von besonderer Bedeutung. Obwohl Linux-HA ab Version 2 die Überwachung von Ressourcen unterstützt, empfiehlt sich externes Monitoring aufgrund der in Kapitel 2.4.3 genannten Kommunikationsstörungen (Split Brain), die innerhalb eines Clusters auftreten können. Dazu verwendet man ein Netzwerk-Management-System (NMS), wie z. B. Nagios [Nag] oder OpenNMS [Opeb]. Das hat weiterhin den Vorteil, dass man nicht die ganze Zeit die Verfügbarkeit der Dienste im Blick haben muss, sondern sich im Fehlerfall per E-Mail oder SMS-Gateway benachrichtigen lassen kann.

NMS greifen in der Regel auf bewährte Standards bei der Überwachung der Dienste zurück. Das am weit verbreitetsten und bekannteste Protokoll zur Überwachung und Management im Netzwerk ist das Simple Network Management Protocol (SNMP). Ein Manager befragt (GET-Request) dabei einen Agenten auf dem zu überwachenden System über den Zustand der Ressource und erhält einen entsprechenden Wert als Antwort (GET-Response). Für eine nicht angeforderte Nachricht vom Agent an den Manager (SNMP-Verwaltungssystem) existiert zudem der SNMP-Trap [Cis]. Diese Art von Nachricht wird von einem sich selbst überwachenden System gesendet, um einen kritischen Zustand mitzuteilen. Das erspart vor allem Prozessor- und Netzwerklast im Gegensatz zum Polling-Verfahren per GET-Request. SNMP ist fast auf allen Netzwerkkomponenten und Hosts integriert - dies ermöglicht die Überwachung der kompletten Netzwerkinfrastruktur über eine zentrale Stelle. Auch Linux-HA liefert einen SNMP-Subagenten mit, sofern dieser mit in das Paket einkompiliert wurde. Andernfalls ist die manuelle Übersetzung von Linux-HA erforderlich.

Da Nagios ohne Plugins nur Statusmonitoring ermöglicht, sollten zusätzliche Tools, wie MRTG [mrt] oder Cacti [cac], zum Verlaufsmonitoring genutzt werden. Damit kann z. B. der Netzwerkverkehr eines Hosts über einen Zeitraum grafisch ausgewertet werden. Daraus lassen sich u. a. Schlussfolgerungen über mögliche Angriffe oder Probleme sonstiger Art treffen. Ferner kann ein Webcrawler die Verfügbarkeit eines Webservers stark beeinträchtigen, sobald dadurch eine Flut von Datenbankanfragen ausgelöst wird. All dies und weitere denkbare Szenarien können durch ein Verlaufsmonitoring erkannt werden, um entsprechende Gegenmaßnahmen ergreifen zu können, die zusätzlich für die Verfügbarkeit eines Dienstes notwendig sind. Darüber hinaus dient das Verlaufsmonitoring zur Dokumentation der in der SLA festgelegten Leistungen gegenüber dem Endkunden.

3.2 Linux-HA

[Sch08] [Sch09] [Kle] [Linb]

3.2.1 Einführung

Das Linux-HA Projekt mit dem zentralen Heartbeat-Dienst ist eine leistungsfähige Clusterlösung für Linux, FreeBSD, Solaris, OpenBSD und zeitweise auch Mac OS X. Die Idee entstand 1997 aus einem Designkonzept von Harald Milz, welcher die Grundlagen und Techniken eines hochverfügbaren Clusters in einem HowTo verfasste. Alan Robertson implementierte auf diesen Ansätzen und Ideen basierend eine Clusterlösung, die als Alpha-Version am 15. November 1998 unter dem Namen Linux-HA veröffentlicht wurde. Bereits im darauf folgenden Jahr wurde Linux-HA beim ersten kommerziellen Kunden eingesetzt, um dessen Internetauftritt hochverfügbar zu machen.

Obwohl die Möglichkeiten der Version 1 (erschienen am 19. Februar 2003) eingeschränkt waren, gab es innerhalb einer kurzen Zeit über 30.000 Installationen. Durch die Veröffentlichung von Linux-HA unter der GPL-Lizenz wurde das Projekt ständig weiterentwickelt und ausgebaut, worauf auch der große Erfolg zurückzuführen ist. So wurden allgemeingültige Lösungen für spezielle Probleme in das Projekt überführt, welche die Qualität und den Nutzen der Software etappenweise stetig verbesserte.

Der in Linux-HA enthaltende Heartbeat-Dienst sorgt für den Nachrichtenaustausch aller beteiligten Knoten, um den Status gegenseitig zu überwachen. Die Ressourcen werden mithilfe eines Cluster Managers (CM) verwaltet und auf den jeweiligen Knoten gestartet. Ressourcen sind im Sinne von Linux-HA all diese, welche vom Cluster verwaltet werden - von IP-Adressen über Dienste, wie einem Webserver, bis hin zu komplexen Mechanismen zur Datenreplikation zwischen den Knoten des Clusters.

Prinzipiell können alle beliebigen Dienste mit Init-Script im System V-Stil hochverfügbar realisiert werden. Darüber hinaus verfügt Linux-HA über spezielle Scripte (Agenten), die eine noch genauere Überwachung, im Gegensatz zu System V-Scripten mit Statusabfrage, ermöglichen.

3.2.2 Linux-HA Version 1

Der Aufbau von Linux-HA Version 1 ist monolithisch. Ein Cluster besteht hierbei aus zwei Knoten, die sich gegenseitig Statusmeldungen (Heartbeat-Pakete) schicken, wodurch auch der Name des Dienstes zustande kommt. Auf dem aktiven Knoten werden alle konfigurierten Ressourcen gestartet, wobei der passive nur im Fehlerfall einspringt, d. h., sobald der Heartbeat ausbleibt. Der passive Node nimmt im diesen Fall an, dass der bisher aktive ein Problem hat und startet dessen Dienste. Dazu zählen z. B. die Cluster IP-Adresse und sonstige Linux-Services, die im Fehlerfall vom Standby-Knoten übernommen werden.

Die Konfiguration eines Linux-HA Version 1 Clusters ist denkbar einfach und vor allem überschaubar, da diese aus nur drei Konfigurationsdateien besteht. In der ersten Datei `authkeys` wird ein gemeinsamer Schlüssel zur Absicherung (MD5, SHA1) der Kommunikation zwischen den Knoten hinterlegt. Die alternative CRC Hash-Methode benötigt dagegen keinen Schlüssel, gilt jedoch als unsicher und sollte daher nicht mehr verwendet werden. Die zweite Datei `ha.cf` enthält Einstellungen, die das Gesamtverhalten des Clusters bestimmen. Dazu zählt z. B. die Angabe beider Knoten des Clusters, Kommunikationseinstellungen sowie die Zeit des Ausbleibens von Statusmeldungen, bis die Reaktion auf den Fehlerfall eintreten soll und damit die Ressourcen durch den passiven Teil des Clusters übernommen werden. In der dritten und letzten Datei `haresources` werden die Ressourcen (mit Parametern) eingetragen, die der Cluster verwalten soll.

Aufgrund des einfachen monolithischen Aufbaus ist die Version 1 für Failover-Cluster auch heute noch beliebt und wird deshalb noch in zahlreichen Systemen eingesetzt. Der Nachteil ist jedoch, dass durch diese statische Architektur nur zwei Knoten in einem Cluster betrieben werden können und die Struktur damit ziemlich unflexibel ist. Die Ressourcen laufen immer auf dem aktiven Knoten, während der passive auf den Fehlerfall wartet, was nicht unbedingt effizient ist, aber in der Welt der Virtualisierung nicht das größte Problem darstellt. Vielmehr fehlt ein Cluster Manager zur Verwaltung der Ressourcen, um diese problemlos auf eine Anzahl von n Knoten zu verteilen. Weiterhin überwacht Linux-HA Version 1 nur den Zustand des Knotens, jedoch nicht den der jeweiligen Ressource, was ein externes Monitoring der Dienste erforderlich macht. Bedingt dadurch, reagiert der Cluster also nicht beim Ausfall eines Dienstes, so dass in Folge dessen dieser z. B. neu gestartet oder im Falle von Speichermangel auf einen anderen Knoten verlagert werden kann. Dafür gibt es eine Reihe zusätzlicher Software, die im Fehlerfall den Heartbeat-Dienst anhält und damit für eine künstliche Übernahme der Ressourcen (Failover) durch Auslösen des Fehlerfalls sorgt.

3.2.3 Linux-HA Version 2

Mit Veröffentlichung von Linux-HA Version 2 am 29. Juli 2005 wurde die Clusterlösung in mehrere Bestandteile (siehe Abbildung 3.1) aufgeteilt. Dabei bekam die Cluster Suite einen eigenen Cluster Manager (CM), wodurch viele Beschränkungen der Vorgängerversion wegfielen und auch neue Funktionen hinzukamen. Mithilfe des neuen Cluster Resource Managers (CRM) ist es nun möglich komplexe Zusammenhänge innerhalb und zwischen den Ressourcen zu definieren, so dass unterschiedliche Ressourcen auch auf unterschiedlichen Knoten im Cluster ausgeführt werden können. Die Hauptaufgabe des CRM besteht in der Ressourcenverwaltung und -überwachung. In Folge dessen ist die externe Überwachung der Ressourcen nicht mehr zwingend nötig, aber dennoch ratsam, da die korrekte Überwachung nicht in jedem Fall, insbesondere bei Kommunikationsstörungen innerhalb des Clusters, gewährleistet werden kann.

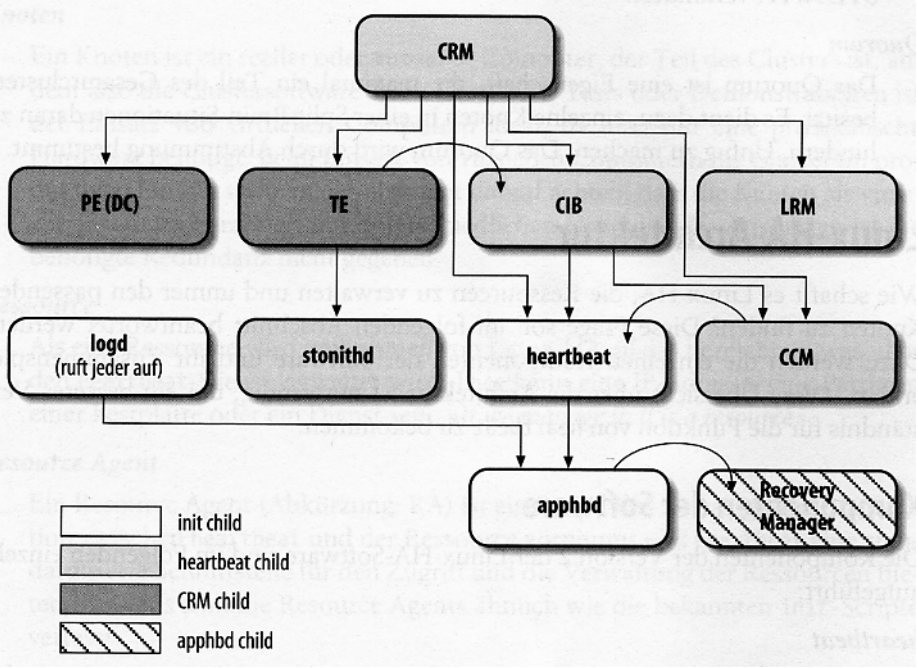

Abbildung 3.1: Komponenten von Linux-HA Version 2 [Sch08]

Der integrierte CRM nutzt für die clusterinterne Kommunikation Heartbeat. Darüber hinaus entscheidet Heartbeat, welche Knoten Mitglieder des Clusters sind. Aufgrund dieser Statusmeldungen sowie der Konfiguration des CRM ist eine optimale Verteilung der Ressourcen über die Nodes möglich. Prinzipiell sind alle Knoten im Cluster gleichberechtigt, solange man durch den CRM keinen Einfluss darauf nimmt, welche Ressourcen auf welchem Knoten laufen und in welcher Reihenfolge diese gestartet und wieder angehalten werden sollen.

Mit Linux-HA Version 2 kam eine Reihe von Funktionalität hinzu, welche nachfolgend kurz dargestellt werden soll:

- Unterstützung bis zu 16 Knoten (M-to-N-Cluster)
- Bei Fehler eines Knotens oder Dienstes wird unter einstellbaren Bedingungen ein Failover ausgelöst
- Unterstützung von Active-/Passive- sowie Active-/Active-Konfigurationen
- Integrierte Ressourcenüberwachung (Agenten) des Clusters und dessen Dienste
- Abhängigkeiten zwischen Ressourcen und Gruppen von Ressourcen konfigurierbar
- Kommunikation zwischen den Knoten per serieller Verbindung über COM oder UDP Broadcast, Multicast und Unicast bei Heartbeat; bei Einsatz von OpenAIS wird generell Multicast verwendet
- Strategien zur Vermeidung von Split Brain-Szenarien; Unterstützung von Quorum und Fencing-Mechanismen
- Administration per GUI oder Kommandozeile über separates Computersystem möglich, welches nicht Mitglied des Clusters sein muss
- Konfigurierbare dynamische Ressourcenverteilung
- Unterstützung verteilter Dateisysteme, wie z. B. OCFS oder GFS

Mit Einführung des neuen CRMs, welcher auch die dynamische Ressourcenverteilung während des Clusterbetriebs unterstützt, wurde die starre Konfiguration mithilfe der Textdatei `haresources` verworfen und die Fortführung erfolgt über eine XML-Datei, welche als Cluster Information Base (CIB) bezeichnet wird und die größte Änderung gegenüber Linux-HA Version 1 darstellt. Der Cluster verwaltet selbst dynamisch die CIB, so dass diese durch den Administrator nicht mehr manuell editiert werden darf. Zur Bearbeitung der CIB existiert eine grafische Benutzeroberfläche (GUI), welche jedoch nur für einfache Bedingungen geeignet ist. Alternativ erfolgt die Bearbeitung über XML-Imports mithilfe der Kommandozeile. Ab Pacemaker Version 1.0.2 existiert eine eigene Shell für die Verwaltung des Clusters. Damit lassen sich z. B. einfacher Attribute und deren Werte für einen ausgewählten Knoten konfigurieren. Dadurch entfällt teilweise die aufwändige manuelle Erstellung einer XML-Datei, die dann in die CIB importiert werden müsste. Für Umsteiger aus Linux-HA Version 1 ist der Aufbau der CIB sehr gewöhnungsbedürftig, da diese einen gewissen Overhead zur Darstellung der Abhängigkeiten besitzt. Der Clusterbau, auf Linux-HA Version 2 basierend, erfolgt damit also größtenteils über die CIB, in der das Verhalten der Ressourcen für den Cluster verständlich hinterlegt werden muss.

Weiterer Projektverlauf

Aus internen Projektstreitigkeiten resultierend, wurde am 7. Dezember 2007 der CRM aus dem Linux-HA Projekt herausgelöst und als eigenständige Komponente unter dem Namen Pacemaker (deutsch: Herzschrittmacher) weitergeführt. Heartbeat 2.1.4 war somit die letzte Version mit integriertem CRM.

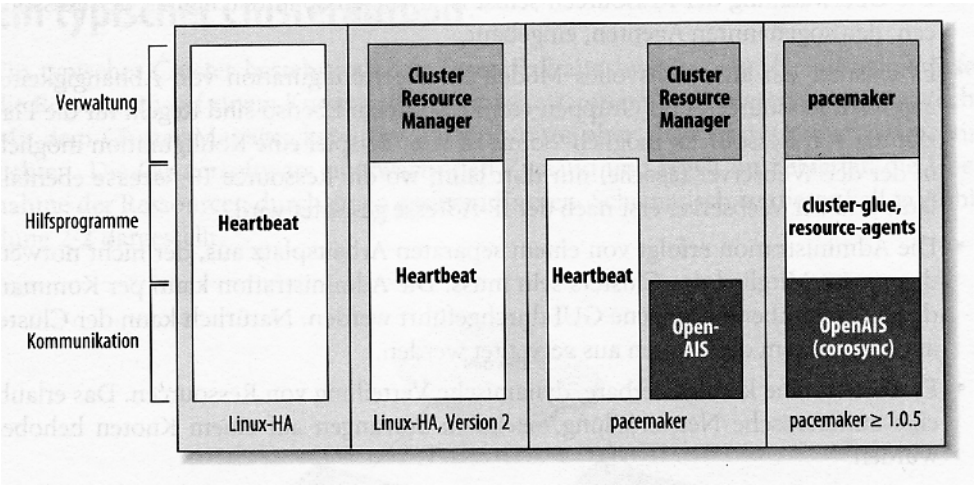

Abbildung 3.2: Historische Entwicklung des Linux-HA Projekts [Sch09]

Der CRM aus Linux-HA Version 2 ist nicht kompatibel zu Pacemaker. Bei Umstellung des CRMs sind somit einige Anpassungen an der Syntax der Cluster Information Base (CIB) erforderlich. Für die Kommunikation im Cluster unterstützt Pacemaker neben Heartbeat nun auch OpenAIS [opea], um Nachrichten zwischen Knoten auszutauschen und sich über die Mitgliedschaft deren im Cluster informieren zu lassen. Der neue CRM Pacemaker lief in dieser Zwischenzeit nur brauchbar mit dem Kommunikationsstack OpenAIS ab Version 0.80.5 zusammen, benötigte dennoch einige Komponenten aus dem Heartbeat-Paket, welche dann das Chaos noch perfektionierten.

Mit Veröffentlichung der Heartbeat-Version 2.99 am 21. August 2008 war Pacemaker nun auch wieder sinnvoll in Verbindung mit Heartbeat als Kommunikationsstack nutzbar. Anzumerken ist hierbei noch, dass einige Versionen von Pacemaker nicht mit bestimmten OpenAIS Versionen zusammenarbeiten. Auf den Projektseiten [pac] findet man entsprechend inkompatible Konstellationen. Speziell in virtuellen Umgebungen kann der Einsatz von Pacemaker in Verbindung mit OpenAIS zu einer CPU-Auslastung von bis zu 50% führen, die durch den `ais`-Prozess verursacht wird. Dieses Problem ist mittlerweile bekannt und wurde in Pacemaker 1.0.5 vollständig behoben.

Abschließend soll noch ein Überblick zur Architektur der Software und den Aufgaben der einzelnen Komponenten (siehe Abbildung 3.1 auf Seite 44) gegeben werden. Für die Infrastruktur zur clusterinternen Kommunikation wird entweder Heartbeat oder OpenAIS (Corosync [cor]) genutzt. Der Cluster Manager verwaltet die Ressourcen auf Basis der Informationen, die er aus der Infrastruktur erhält. Für die Verwaltung ist mittlerweile der Projektnachfolger Pacemaker zuständig. Dieser ist die zentrale Instanz, die entscheidet, welche Ressourcen auf welchem Knoten des Clusters laufen sollen. Die beiden Kommunikationsstacks Heartbeat oder OpenAIS liefern Informationen über den Zustand der Knoten, während die Konfiguration des Administrators bestimmte Bedingungen vorgibt. Aus all diesen Informationen berechnet Pacemaker die optimale Verteilung der Ressourcen über ein Punktesystem und vergleicht dabei SOLL- und IST-Zustand. Dazu nutzt Pacemaker eine Reihe von Diensten, die nachfolgend kurz vorgestellt werden sollen und aus [Sch09] entnommen wurden:

- Cluster Resource Manager (CRM)
  Der CRM verwaltet die Konfiguration der Ressourcen, dieser entscheidet, welche Ressource, wo laufen soll, und wie der gewünschte Zustand, vom aktuellen ausgehend, am besten erreicht werden kann. Weiterhin überwacht der CRM den Local Resource Manager, wie die Vorgaben auf den jeweiligen Knoten umgesetzt werden. Der CRM
  - nutzt Heartbeat oder OpenAIS für die Kommunikation.
  - erhält Meldungen über den Zustand der Mitgliedschaft von Knoten vom Consensus Cluster Manager (CCM) oder von CLM bei OpenAIS.
  - verteilt die Arbeit an den Local Resource Manager (LRM) und erhält Rückmeldung über die Umsetzung der Aufgaben.
  - teilt dem STONITH-Dienst mit, welche Prozesse zu welcher Zeit neu zu starten sind.
  - loggt die Aktivitäten mithilfe des Logging-Dienstes.
- Policy Engine (PE)
  Die Policy Engine berechnet den Übergang zwischen aktuellem Zustand und möglichem Idealzustand des Clusters. Dabei wird die momentane Verteilung und Zustand der Ressourcen, die Verfügbarkeit der Knoten sowie die vorgegebene Konfiguration berücksichtigt. Das Ergebnis der Berechnung im XML-Format dient als Eingabe für eine Transition Engine (TE), die letztendlich den Übergang durchführt, indem diese den LRM auf den betroffenen Knoten anweist, bestimmte Ressourcen zu starten oder zu stoppen.

- Cluster Information Base (CIB)
  Die CIB ist der zentrale Informationsspeicher (XML-basiert) des Clusters über Ressourcen und Knoten, welche dem CRM zur Verfügung gestellt und auf alle Knoten automatisch repliziert wird. Darin sind statische Konfigurationen (z. B. Abhängigkeiten) sowie dynamische Informationen (z. B. aktueller Zustand des Knotens oder einer Ressource) enthalten.
- Cluster Abstraction Layer
  Diese Abstraktionsschicht zwischen CRM und der Kommunikationsebene dient zur Verallgemeinerung der Meldungen für den CRM, so dass der Einsatz des Kommunikationsstack für die clusterinterne Kommunikation flexibel bleibt.
- Consensus Cluster Membership (CCM)
  Die Consensus Cluster Membership (CCM) berechnet die Mitgliedschaft eines Knotens zum Cluster und sorgt dafür, dass die Knoten innerhalb eines Clusters untereinander kommunizieren können. Diese Komponente ist in Heartbeat weit gehend selbstständig - in OpenAIS hingegen ist diese im CLM integriert.
- Local Resource Manager (LRM)
  Der Local Resource Manager (LRM) ist eine Abstraktionsebene oberhalb der jeweiligen Resource Agents auf den Knoten. Der LRM ist für die Durchführung von Start- und Stoppvorgängen zuständig und überwacht die Ressourcen, wie vom CRM vorgegeben.
  Ferner unterstützt dieser dabei folgende Klassen von Ressourcen:
  - OCF: Open Cluster Framework
  - Agenten, die dem Heartbeat Version 1 Standard entsprechen
  - LSB: Linux Standard Base (Init-Scripte im System V-Stil)
  - STONITH Agenten als eigene Klasse
- STONITH-Dienst
  Der STONITH-Dienst kann einzelne Knoten des Gesamtclusters abschalten, um eine Split Brain-Situation zu vermeiden.

Einige Hintergrunddienste (Non-Blocking Logging Daemon, Infrastruktur sowie Cluster Testing System) werden an dieser Stelle nicht näher betrachtet, da diese nicht von Relevanz für die vorliegende Diplomarbeit sind. Nähere Informationen darüber findet man in [Sch09]. Außerdem wird dort das Zusammenwirken der Komponenten erläutert.

3.2.4 Linux-HA Version 3 und zukünftige Entwicklung

Am 18. Januar 2010 erschien Linux-HA Version 3, in der die Paketaufteilung geändert wurde und den Heartbeat-Dienst nur noch bei entsprechender Konfiguration in Verbindung mit Pacemaker für Kommunikationszwecke nutzt. Die neue Paketstruktur erlaubt somit den Betrieb von modernen Pacemaker-Clustern mit OpenAIS ohne irgendwelche Bestandteile von Heartbeat mit installieren zu müssen. Bisher waren einige Verbindungskomponenten zwischen der Verwaltung und der binären Programme der Ressourcen (Agenten) in Heartbeat verblieben.

Gemeinsam mit der Veröffentlichung von Pacemaker 1.0.5 wurde der verbliebene Bestandteil von Linux-HA wie folgt aufgeteilt:

- `resource-agents`
  Dieses Paket beinhaltet alle Agenten, welche das Bindeglied zwischen Clustersoftware und den binären Programmen, die der Cluster ausführen soll, darstellen. Vergleichbar sind diese mit den Scripts, die das Init-System benutzt, bieten jedoch in der Regel mehr Möglichkeiten, vor allem zur Überwachung der jeweiligen Dienste.
- `cluster-glue`
  In diesem Paket sind alle restlichen Bestandteile aus dem ursprünglichen Heartbeat-Paket zu finden, die auch noch für den Betrieb von Pacemaker in Verbindung mit OpenAIS notwendig sind.
- `heartbeat`
  Ab Version 3 von Heartbeat wurde der verbliebene Rest der ursprünglichen Software zusammengefasst, die nicht von den anderen beiden Paketen benötigt wird und dem Funktionsumfang eines Cluster Managers von Linux-HA Version 1 entspricht.

Dieser Schritt war längst überfällig, um dem bisherigen Chaos ein Ende zu bereiten. Damit wird letztendlich auch das Ende von Heartbeat für die Kommunikation im Clustersystem eingeleitet, da OpenAIS dieses Problem grundsätzlicher löst. Der Trend geht deshalb ganz klar in Richtung Pacemaker in Verbindung Corosync, einer abgespeckten Variante von OpenAIS. Die treibende Kraft hinter dem CRM Pacemaker ist Novell, und die moderne Variante der Kommunikationsplattform OpenAIS kommt größtenteils aus dem Hause Red Hat. Da Red Hat bereits eine eigene Cluster Suite pflegt, die für die Kommunikation ebenso OpenAIS nutzt, ist eine Kooperation der Projekte zukünftig geplant. Weiterhin arbeitet ein Großteil der Entwickler für die bereits genannten Linux-Distributionen. Weitere große Unterstützung erfährt das Projekt von der Firma LINBIT [LINa], dem Hersteller von DRBD [LINe]. Durch die Modularisierung (Trennung zwischen Cluster Manager und Cluster-Kommunikation) könnten sich in Zukunft weitere Lösungen etablieren.

Allgemein werden unter Linux-HA Version 3 moderne Pacemaker-Cluster mit OpenAIS bzw. Corosync verstanden, was jedoch falsch ist. Die Pacemaker-Entwickler haben bei der Aufteilung des Heartbeart-Projekts eine ablehnende Haltung gegenüber dem Begriff Linux-HA entwickelt und versuchen diesen zu meiden, wo es nur möglich ist. Außerdem wurden mit Veröffentlichung von Pacemaker 1.0.5 die letzten Überreste von Heartbeat aus dem Code-Repository von Pacemaker entfernt. Das Paket `cluster-glue` beinhaltet nun den Local Resource Manager, welcher die Schnittstelle zu den Agenten (OCF, LSB und STONITH) bildet, wohingegen die Agenten selbst in das Paket `resource-agents` gewandert sind. Genauer betrachtet ist Linux-HA Version 3 eigentlich der Heartbeat-Dienst Version 1 mit den abgespaltenen Paketen für den Betrieb von Pacemaker-Clustern. Diese Paketstruktur wird man zukünftig auch in den Linux-Distributionen wiederfinden. Der Begriff Linux-HA wird dennoch aus Marketinggründen erhalten bleiben, da dieser das Projekt am treffendsten beschreibt und auch jedem bekannt ist.

3.2.5 Resümee

Aufgrund des komplexen Aufbaus und der bisher unklaren Projektstruktur wird für die HA-Konfigurationen dieser Diplomarbeit auf Linux-HA Version 2 mit CRM verzichtet. Kritische Dienste lassen sich mit einem Failover-Cluster auch auf Basis von Linux-HA Version 1 absichern. Das hat den Vorteil, dass die Konfiguration überschaubar bleibt und gemäß dem KISS-Prinzip erfolgt. Ferner steigt die Komplexität eines M-to-N-Clusters mit steigender Knotenanzahl ins Unermessliche, so dass bei Ausfall einzelner Knoten das Verhalten des Clusters unter Umständen nicht mehr eindeutig vorhersehbar ist. Auch bei Wiederinbetriebnahme eines defekten Knotens ergibt sich eine ähnliche Situation. Kombiniert man weiterhin die Möglichkeiten der Virtualisierungssoftware, insbesondere die dynamische Live-Migration von VMs, mit der dynamischen Ressourcenverteilung der Cluster Suite, wird das System unbeherrschbar, sofern der CRM keine Kenntnis davon hat, dass es sich hierbei um VMs handelt.

Das Linux-HA Projekt ist sehr dynamisch und stetig im Wandel, wie auf der Projektseite [Kle] momentan erkennbar ist. Es erfolgt die Migration der bisher völlig unstrukturierten Projektseite in ein Wiki [Linb]. Des Weiteren ist die Dokumentation zu Linux-HA Version 2 sehr spärlich gehalten, was die Konfiguration dieser vielschichtigen Clusterlösung erheblich erschwert. Der Artikel “Hochverfügbarkeit mit Linux im Wandel” des Linux-Magazins von Martin Loschwitz [Los09] unterstreicht die gewonnen Erkenntnisse noch einmal deutlich und fasst diese übersichtlich zusammen.

3.3 Linux-HA Partnerprojekte

3.3.1 Distributed Replicated Block Device

[LINe] [Bre07] [Sch09]

Distributed Replicated Block Device (DRBD) ist eine OpenSource-Lösung der Firma LINBIT [LINa] zur gemeinsamen Datenhaltung von Blockgeräten über das Netzwerk. DRBD ist vor allem für einen Failover-Cluster eine kostengünstige Alternative zu einem Shared Storage. Es ermöglicht die Replikation von Daten auf Blockgeräteebene, d. h. eine Art RAID 1 der Festplatte (HDD) des Clusterknotens über das Netzwerk.

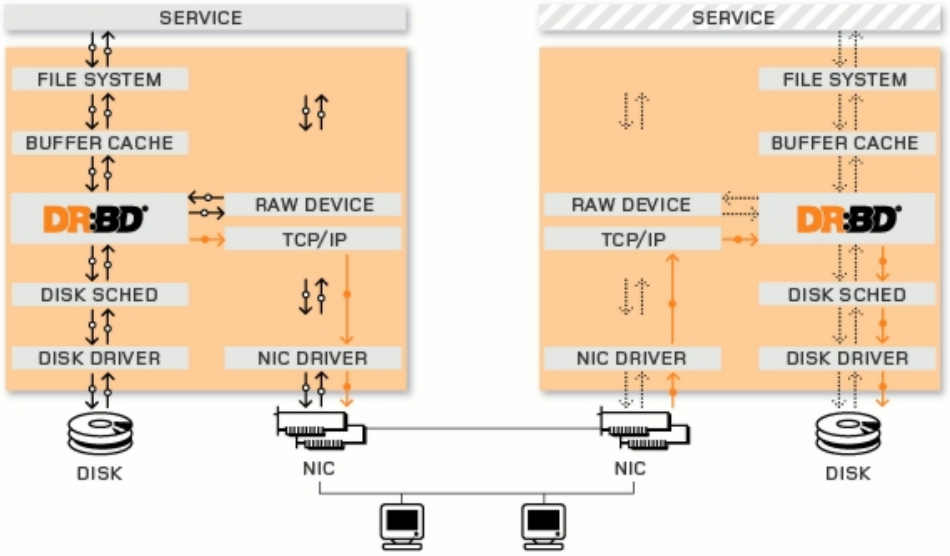

Abbildung 3.3: Funktionsweise DRBD [LINe]

Im Regelfall läuft DRBD als Master-/Slave-Konfiguration zwischen zwei Knoten. Der Einsatz eines herkömmlichen Linux-Dateisystems, wie z. B. ext3, ext4 oder XFS, ist für diese Variante empfehlenswert, da im Normalfall nur ein Knoten gleichzeitig darauf zugreifen kann. Alle Schreibzugriffe werden an den passiven Knoten des Clusters über das Netzwerk transportiert. Erst wenn der Schreibvorgang erfolgreich auf dem passiven Knoten durchgeführt wurde, erfährt die entsprechende Applikation davon. Dieses Verfahren entspricht dem Replikationsmodus C (synchroner Datentransfer) in DRBD, der für die meisten Clusterkonfigurationen genutzt werden sollte, da hier kein Datenverlust auftreten kann. Replikationsmodus A (asynchroner Datentransfer) meldet hingegen den Schreibvorgang als erfolgreich, sobald die Operation auf dem aktiven Knoten abgeschlossen wurde und repliziert erst danach die Daten. Weiterhin existiert Replikationsmodus B (semi-synchronous), der ähnlich wie Modus C, jedoch mit Caching der zu schreibenden Daten

im RAM, arbeitet. Die Wahl des geeigneten Replikationsmodus hängt letztendlich von der Netzwerklatenz mit Auswirkung auf die Schreibgeschwindigkeit und des Bedarfs an die Datenkonsistenz ab. Synchron sollten Daten repliziert werden, die in jedem Fall konsistent sein müssen, wohingegen temporäre Daten je nach Anforderung auch asynchron gespiegelt werden können. Letztlich ist dies jedoch wieder vom individuellen Anwendungsfall abhängig.

DRBD erzeugt ein virtuelles Gerät zur Datenreplikation, das die Verbindung zur logischen Partition auf die HDD herstellt, für die die Datensynchronisation konfiguriert wurde. Die logische Partition kann nicht mehr direkt angesprochen werden, so dass der Zugriff auf die Daten nun über das virtuelle Gerät erfolgt. Außerdem besitzt dies auf dem jeweiligen Knoten einen Primär- bzw. Sekundärstatus. Im Fehlerfall versetzt der Cluster Resource Manager das sekundäre Gerät in den Primärzustand. Während der Übernahme des Zustandes ist eine Dateisystemintegritätsprüfung erforderlich, welche durch eine Dateisystemprüfung oder durch das Zurückspielen des Dateisystemjournals erfolgt. Sobald das bisherige primäre System wieder verfügbar ist, erfolgt eine Resynchronisation des Datenbestandes, welcher im Anschluss daran wieder in den Primärmodus versetzt wird. Es muss hierbei nicht das gesamte Blockgerät synchronisiert werden, sondern nur der Teil der Datenblöcke, die während des Ausfalls geändert wurden.

Zur Installation von DRBD musste bisher das Kernelmodul selbst kompiliert werden. Das Projekt schaffte jedoch vor kurzem die Aufnahme in den Linux Kernel 2.6.33, der als Stable Release im ersten Quartal 2010 erscheinen wird. Dadurch muss nur noch die Management-Applikation im Userspace zum Kernelmodul selbst installiert werden. Zur Integration in Linux-HA Version 1 (Heartbeat) existiert ein Resource Agent, allerdings steht erst seit DRBD Version 8.3.2 ein OCF kompatibler Heartbeat/Pacemaker Resource Agent zur Verfügung. Das bisher kommerzielle DRBD+ wurde im Dezember 2008 mit der OpenSource-Variante zusammengeführt. Daraufhin wurde der Funktionsumfang in DRBD Version 8 u. a. um folgende erweitert:

- Unterstützung von Dual Primary Mode für den gleichzeitigen Zugriff auf den Datenbestand (Shared Storage) mithilfe eines verteilten Dateisystems, wie z. B. GFS oder OCFS2 (derzeit nicht für den produktiven Einsatz empfohlen)
- 3-Wege-Replikation für Backup- oder Disaster Recovery-Zwecke
- Unterstützung von Blockgerätgrößen bis zu 16 TB
- zahlreiche Performanceoptimierungen für den WAN-Betrieb

DRBD gilt als zuverlässige Replikationslösung für Clustersysteme. Durch den Einsatz intelligenter Algorithmen wird eine hohe Performance und Sicherheit gewährleistet.

3.3.2 Linux Virtual Server

[Zhac] [Bre07] [Sch09]

Linux Virtual Server (LVS) [Zhac] (siehe Abbildung 3.4) ist eine Software zur Lastverteilung (Load Balancing) von TCP- bzw. UDP-Verbindungen auf Basis von OSI Layer 3 und 4. Die Software ist schon seit längerem fester Bestandteil des Linux 2.4 Kernel- und Linux 2.6 Kernelzweiges und gilt damit als ausgereifte robuste Lösung, auf der mittlerweile mehrere Projekte zur dynamischen Verwaltung der Weiterleitungstabelle des LVS aufbauen. Durch die direkte Kernelintegration entfällt das aufwändige Context-Switching sowie das Kopieren von Speicher zwischen User- und Kernel-Mode.

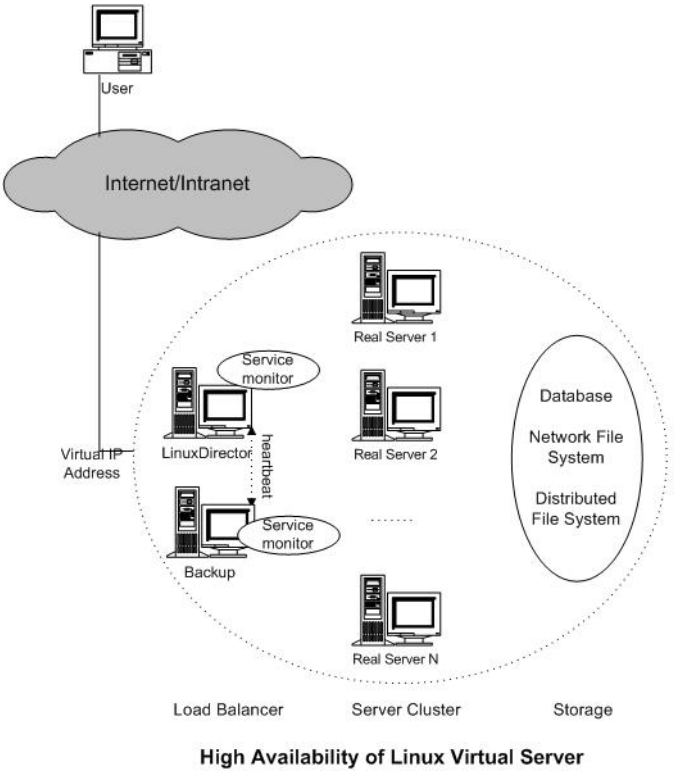

Abbildung 3.4: Übersicht Linux Virtual Server [Zhac]

LVS ermöglicht Server Load Balancing (SLB), so dass mehrere physische Hosts (sog. Real Server) unter einer gemeinsamen virtuellen IP-Adresse (VIP) erreichbar sind. Der Linux Director als Load Balancer richtet dafür eine VIP als zusätzliche IP-Adresse für das externe Interface ein, unter der dann die gesamte Serverfarm erreichbar ist, und nimmt damit alle Client-Anfragen an die VIP entgegen und leitet diese nach eingestelltem Scheduling-Algorithmus an einem Server im Netzwerk weiter. Damit können u. a. mehrere parallele Verbindungen des selben Clients auf unterschiedliche Real Server weitergeleitet werden. Bedingt durch die Arbeitsweise auf Layer 3 und 4 kann der Linux Director

diese Entscheidungen ausschließlich anhand der IP-Adresse und der jeweils konfigurierten TCP- oder UDP-Ports treffen. Der Inhalt des Paketes oder das zuständige Übertragungsprotokoll wird hierbei nicht weiter vom Director berücksichtigt. Dementsprechend erreicht man mit dieser Art von SLB eine hohe Performance, da nur wenige Headerinformationen der Pakete berücksichtigt werden müssen.

Zur logischen Verteilung der eingehenden Verbindungen existieren in LVS folgende verschiedene Scheduling-Algorithmen:

- Round Robin Scheduling
- Weighted Round Robin Scheduling
- Least Connection Scheduling
- Weighted Least Connection Scheduling
- Locality Based Least Connection Scheduling
- Locality Based Least Connection with Replication Scheduling
- Destination Hashing Scheduling
- Source Hashing Scheduling
- Shortest Expected Delay Scheduling
- Never Queue Scheduling

Die meist verwendeten Varianten sind (Weighted) Round Robin sowie (Weighted) Least Connection. Round Robin (RR) ist die einfachste Variante und verteilt die Verbindungen reihum auf die Server, ohne dabei zusätzliche Parameter zu berücksichtigen. Diese Methode ist bei homogener Serverhardware und für die Lastverteilung mit relativ konstanten Bearbeitungszeiten (z. B. statische Websites) sinnvoll. Eine weitere Vorgehensweise ist Least Connections (LC), bei der der LB die Tabelle der momentan bestehenden Verbindungen berücksichtigt und den Server mit der geringsten Aktivität auswählt. LC eignet sich vor allem bei deutlich unterschiedlichen Bearbeitungszeiten, z. B. im Webbereich bei Downloads oder für die Verarbeitung umfangreicher dynamischer Seiten. Bei unterschiedlich performanter Serverhardware ist LC im Vergleich zu RR von Vorteil, da leistungsstärkere Maschinen Verbindungen schneller abarbeiten und dadurch automatisch zusätzliche Anfragen zugeteilt bekommen. Um Einfluss auf die gewählte Strategie zu nehmen, existieren die Weigthed-Varianten zur Gewichtung der einzelnen Server. Locality Based Least Connection Scheduling (LBLC) weist einem Client im Regelfall immer denselben Server zu, sofern dieser nicht nach eingestellter Gewichtung als überlastet gilt. Weitere Details zu den nicht näher erläuterten Verfahren findet man unter [Zhaa].

Zur technischen Verteilung der Pakete an die Server verfügt LVS über folgende Verfahren:

- Network Address Translation (NAT)
  In dieser Betriebsart verhält sich der Director ähnlich wie ein NAT-Router, der die Zieladressen der Pakete umsetzt und diese dann weiterleitet. Die Quell-IP bleibt unverändert. Der jeweilige Real Server hat den Director somit auch als Default Gateway eingetragen, damit die Quell-IP des Real Servers wieder in die öffentliche IP des LB umgesetzt werden kann. Die Zuordnung von Port und Server erfolgt dabei nicht statisch, sondern anhand der eingestellten Scheduling-Verfahren.
  Diese Methode eignet sich vor allem in Netzwerktopologien, indem sich der Load Balancer mit jeweils einem Netzwerkinterface in einem öffentlichen und einem internen Serversubnetz befindet. Das hat den Vorteil, dass auf der Serverseite jedes beliebige TCP/IP-fähige Betriebssystem betrieben werden kann und sich die Real Server in einem abgetrennten privaten Subnetz befinden. Weiterhin benötigt man nur eine öffentliche IP-Adresse für den Linux Director und auch die Integration in eine bestehende Netzwerkinfrastruktur ist damit am einfachsten möglich, da die Real Server an irgendeinem beliebigen Ort stehen können. Der Nachteil an diesem Verfahren ist die schlechte Skalierbarkeit, da alle Pakete (Request und Response) über den LB geroutet werden müssen und dieser damit zum Flaschenhals wird. Außerdem müssen natürlich die NAT-Tabellen gepflegt werden. Üblich sind je nach Leistungsfähigkeit des LB Konfigurationen von bis zu 30 Real Servern.

- IP-Tunneling mit Encapsulation
  Beim IP-Tunneling nutzt der Linux Director ein IP-Tunnel zum jeweils ausgewählten Real Server, indem die Pakete mit einem neuen IP-Header mit der Ziel-IP vor dem ursprünglichen Header versieht. Das Verpacken (Encapsulation) belässt den Original-Header unverändert - die Ziel-IP bleibt die VIP des LB. Der Real Server macht die Encapsulation rückgängig und beantwortet die Anfrage mit der VIP als Quell-IP. Das ist jedoch nur möglich, sofern die Konfiguration der VIP als Alias auf dem jeweiligen Netzwerkinterface konfiguriert ist. Dadurch wird die tatsächliche Adresse des Clients auf diesem Weg nicht geändert, das dem jeweiligen Real Server ermöglicht dem Client direkt zu antworten, so dass damit das Paket nicht mehr über den LB, sondern auch über einen regulären Router zur Entlastung des LB geroutet werden kann. Bedingt dadurch kann die Netzwerktopologie frei gewählt werden, so dass auch eine geografische Verteilung der Server denkbar ist.
  Der Vorteil von diesem Verfahren liegt in der besseren Skalierbarkeit gegenüber NAT für größere Konfigurationen von 100 und mehr Real Servern. Die Requests sind oft eher klein, dagegen fallen die Replys größer aus - was in diesem Betriebs-

modus den Director extrem entlastet. Das erlaubt diesem viele Requests gleichzeitig zu verarbeiten, ohne sich dabei um die Antwort kümmern zu müssen. Der Nachteil hingegen ist, dass nicht mehr jedes beliebige Betriebssystem verwendet werden kann, da IP-Encapsulation durch den Netzwerkstack unterstützt werden muss. Weiterhin muss gewährleistet werden, dass nur der LB auf ARP-Requests für die VIP antwortet, sofern sich dieser im selben Netzsegment wie die Real Server befindet.

- Direct Routing (MAC Address Translation)
  Direct Routing (DR) basiert auf der Idee vom IP-Tunneling zur Entlastung des LB, verzichtet hingegen auf die Encapsulation der Antwortpakete. Beim DR müssen sich Linux Director und die Real Server im selben physikalischen Netzsegment befinden, d. h., alle sind über einen Switch verbunden. Sowohl LB als auch die Real Server müssen in diesem Betriebsmodus die VIP als Alias auf dem jeweiligen Netzwerkinterface konfiguriert haben. Wichtig ist hierbei, dass die Real Server nicht auf ARP-Requests für die VIP antworten, was sonst zu einem Chaos in diesem lokalen Netzsegment führen würde.
  Die Funktionsweise des DR ist denkbar einfach. Sobald der LB eine Anfrage erhält, wird das Paket auf Layer 2 so geändert, dass die Ziel-MAC des ausgewählten Real Servers an der entsprechenden Stelle ausgetauscht und das Paket einfach ins Netzsegment weitergeleitet wird. Der Real Server bemerkt von dieser Änderung nichts und schickt das Antwortpaket ohne Umweg mit der VIP als Quell-IP direkt an den Client. Die Paketmanipulation auf Layer 2 ist sehr ressourcenschonend, was sich auch an der deutlich besseren Performance bemerkbar macht. Der Vorteil von DR ist hiermit die Skalierbarkeit für größere Konfigurationen, in der auch 100 und mehr Real Server kein Problem darstellen. Ferner ist der geringe Speicherbedarf von LVS (pro Verbindung 128 Byte in der Hash Table) positiv hervorzuheben, so dass auch eine hohe Anzahl an parallelen Verbindungen möglich sind. Der Nachteil bleibt dennoch, dass LB und Real Server unter Umständen im gleichen öffentlichen Netzsegment sein müssen.

Zusammenfassung

Um Missverständnisse zu vermeiden, soll an dieser Stelle noch einmal ausdrücklich darauf hingewiesen werden, dass sich mit LVS die Last auf viele statische Server optimal verteilen lässt. Allerdings verfügt das `ip_vs`-Kernelmodul nicht über Monitoring-Funktionen, ob die eingetragenen Server bzw. deren angebotene Dienste noch ordnungsgemäß ihre Arbeit verrichten. Dabei wird nur die Grundfunktionalität zur statischen Verwaltung der zur Verfügung stehenden Server bereitgestellt. Für ein solch hoch skalierbares Serversystem ist ein Zusatzprozess notwendig, welcher die Weiterleitungstabellen von LVS dynamisch je nach Verfügbarkeit der konfigurierten Server verwaltet. Ein Beispiel dafür ist `ldirectord`, welcher im folgenden Kapitel detailliert behandelt wird.

Da der Director einen SPOF darstellt, ist dieser durch eine Clusterlösung in Form eines Failover-Clusters abzusichern, damit auch bei Ausfall eines Directors der störungsfreie Betrieb weiterhin gewährleistet werden kann. Dazu ist u. a. die Übernahme der VIP im Fehlerfall sowie die permanente Synchronisation der Verbindungstabellen des LVS nötig. Dafür existieren mehrere Resource Agenten in Linux-HA Version 1. Auch mit Pacemaker lässt sich `ldirectord` ohne Probleme hochverfügbar machen, da dafür ein OCF Resource Agent vorhanden ist.

3.3.3 Ultra Monkey

Ultra Monkey [Zhad] ist ein Projekt um lastverteilte und hochverfügbare Dienste in Verbindung mit LVS zu realisieren. Der darin enthaltende Prozess `ldirectord` kann verschiedene Dienste auf Verfügbarkeit prüfen. Dies reicht vom einfachen regelmäßigen Pingen aller Server bis hin zur Durchführung von MySQL-Queries. Es können dabei folgende Dienste per Request und Response geprüft werden: HTTP, HTTPS, FTP, IMAP, POP, SMTP, LDAP, NNTP und MySQL - für alle weiteren Services steht ein einfacher Portcheck zur Verfügung.

Auf Basis dieser Informationen kann `ldirectord` die Weiterleitungstabelle des LVS dynamisch pflegen. Im Falle eines Webservers bietet `ldirectord` die Möglichkeit per GET-Request eine konfigurierte Testseite mit einem definierten Inhalt abzurufen. Sofern die Testseite ordnungsgemäß abgerufen wird und die Zeichenkette übereinstimmt, bleibt der jeweilige Webserver in der Weiterleitungstabelle. Der Dienst hat eine sehr einfache Konfigurationsdatei in der alle benötigten Parameter für LVS eingestellt werden. Darin werden u. a. die VIP, die IP der Real Server sowie der Modus zur Paketweiterleitung und des Scheduling-Algorithmus konfiguriert.

Für `ldirectord` existiert weiterhin ein Resource Agent zur Einbindung in Heartbeat. Das ist insofern sinnvoll, dass nicht ständig beide LB die konfigurierten Dienste auf Verfügbarkeit prüfen, da ein Abgleich der Verbindungstabellen in jedem Fall über weiteren Dienst (LVS State Sync Daemon) in Form eines Resource Agents erfolgt.

Weitere ähnliche Lösungen sind Piranha [Redb] von Red Hat oder Keepalived [kee], das auf Basis des Virtual Router Redundancy Protocol (VRRP) arbeitet.

3.3.4 Oracle Cluster File System 2

Kein direkter Bestandteil des Linux-HA Projektes oder Projektpartner ist das Oracle Cluster File System 2 (OCFS2) [Ora], welches zu den verteilten Dateisystemen gehört. Da jedoch in Verbindung mit hochverfügbaren Clusterlösungen auch verteilte Dateisysteme benötigt werden, erfolgt an dieser Stelle ein kurzer Überblick. Insbesondere wird OCFS2 auch zur Einbindung in Linux-HA ab Version 2 unterstützt. Dadurch entfallen die Konfigurationsdateien auf den einzelnen Knoten und das aufwändige manuelle Abgleichen bei Änderungen, da die Konfiguration komplett vom Distributed Lock Manager (DLM) sowie OpenAIS übernommen wird. Anzumerken ist hierbei, dass es an dieser Stelle nicht genügt, die abgespeckte OpenAIS Variante Corosync zu installieren, sondern immer noch zusätzlich OpenAIS notwendig ist, da der DLM OpenAIS in dieser Konfiguration für die clusterinterne Kommunikation nutzt.

OCFS2 ist ein OpenSource Cluster Dateisystem und seit Linux-Kernel 2.6.16 direkt integriert. Weiterhin spricht die einfache Konfiguration für die Verwendung von OCFS2. Bedingt durch die parallelen Zugriffe mehrerer Knoten bei verteilten Dateisystemen auf dieselben Datenblöcke, ist das Locking dieser Bereiche nötig, um Inkonsistenzen zu vermeiden. Dazu verwendet OCFS2 einen DLM, der die folgenden Lock-Modi unterstützt: EXMODE (Exclusive), PRMODE (Read only) und NLMODE (No Lock).

Da sich die vorliegende Diplomarbeit nicht näher mit verteilten Dateisystemen beschäftigt, wird an dieser Stelle auf weitere Details verzichtet. Auf den Oracle Projektseiten findet man unter [Ora] und [Fas] weitere Informationen zum internen Aufbau und der Funktionsweise von OCFS2.

Weitere bekannte verteilte Dateisysteme sind, z. B. das von Red Hat entwickelte Global File System (GFS), Lustre von Sun Microsystems oder das General Parallel Filesystem (GPFS) von IBM. Einen anderen alternativen Ansatz für die gemeinsame Datenhaltung im Cluster ermöglicht der clusterfähige Logical Volume Manager (CLVM) [Reda].

3.3.5 Csync2

[Bre07]

Csync2 [LINf] ist ein OpenSource-Projekt der Firma LINBIT [LINa] zur Synchronisation von Dateien in einem Clustersystem. Das ist vor allem bei der Änderungen von Konfigurationsdateien im Cluster hilfreich, da oft vergessen wird, diese auf alle Knoten zu transferieren und damit auch eine Funktionsstörung des Clusters verursacht werden kann. Alternativ werden oftmals Linux Tools, wie rsync oder scp, in Shell Scripten genutzt. Diese stoßen aber schnell an ihre Grenzen. Die Lösung dafür bietet Csync2. Es unterstützt die Synchronisation von Löschvorgängen, indem es die Informationen in einer Datenbank vorhält. Somit ist es möglich im Nachhinein festzustellen, ob eine Datei auf einem Knoten neu angelegt oder gelöscht wurde. Des Weiteren werden Konflikte in Umgebungen mit mehreren Administratoren erkannt, die möglicherweise eine bestimmte Datei auf einem Knoten geändert, aber keine Synchronisation vorgenommen haben. Sobald nun genau diese Datei auf einem anderen Knoten bearbeitet und eine Synchronisation vorgenommen wird, erkennt Csync2 den Konflikt und meldet einen Fehler. Selbst der Einsatz in komplexen Clustersystemen stellt kein Problem dar, da durch eine zentrale Konfigurationsdatei auf allen Knoten auch einzelne Dateien von der Synchronisation auf bestimmte Nodes ausgenommen werden können. Außerdem unterstützt Csync2 das Ausführen von Aktionen, sofern bestimmte Konfigurationsdateien geändert wurden. Sobald z. B. die Konfiguration des FTP-Servers geändert wurde, kann dieser neu gestartet werden, um die Änderung der Konfiguration zu übernehmen. Auch die Sicherheit ist beim Transfer der Daten gewährleistet, da die Übertragung mit SSL/TLS verschlüsselt wird.

Die Konfiguration von Csync2 ist auf der Projektseite unter [LINf] oder in [Bre07] weitreichend erklärt.

Kapitel 4

HA-Konfigurationen

4.1 Überblick

Um die in Kapitel 3 ausführlich dargestellten Hochverfügbarkeits-Implementierungen an praktischen Beispielen umzusetzen, wurde bei der Auswahl dieser auf grundlegende Prinzipien geachtet. Im Vordergrund steht hierbei vor allem die Übertragbarkeit auf ähnliche Anwendungsfälle, damit die Dienste hierfür hochverfügbar realisiert werden können. Zum einen sollen kostengünstige Clusterlösungen ohne externes Storage-System und zum anderen jedoch auch umfangreichere Konfigurationen unter Verwendung eines verteilten Dateisystems demonstriert werden, um dadurch einen Großteil aller praxisrelevanten Serverdienste in einer hochverfügbaren Umgebung abzudecken. Es wurde weiterhin darauf geachtet, den Konfigurationsaufwand möglichst gering zu halten, um den Schwerpunkt der Diplomarbeit entsprechend auf Hochverfügbarkeit und Virtualisierung auszurichten.

Das erste ausgewählte Beispiel zeigt einen **FTP-HA-Cluster**, welcher aus zwei Knoten besteht, die sich gegenseitig per Heartbeat überwachen. Es dient vorläufig zum Einstieg des Clusterns von Serversystemen ohne zentraler Datenhaltung. Dazu sollen die grundlegenden Funktionen eines Failover-Clusters unter Einsatz von Linux-HA Version 1 vorgestellt werden. Für den gemeinsamen Datenbestand sorgt ein synchron repliziertes Blockdevice, welches an an dieser Stelle sogar noch einen SPOF des Storage-Systems beseitigt. Auf Basis dieser Beispielkonfiguration können weitere bekannte Dienste, wie z. B. NFS oder SMB, hochverfügbar realisiert werden.

Die zweite Beispielkonfiguration umfasst einen **Webserver-HA-Cluster mit Load Balancer**, welcher das grundlegende Konzept der ersten Lösung erweitert. Unter Verwendung von zwei Software Load Balancern, die sich wiederum gegenseitig per Heartbeat überwachen, wird ein hochverfügbarer sowie hoch skalierbarer Webcluster aufgebaut. Durch eine Erweiterung des LB können die Webserver überwacht und dadurch deren

Verfügbarkeit periodisch abgefragt werden. Auf Grundlage dieser Statusinformationen pflegt der LB seine Weiterleitungstabelle dynamisch. Da bei einigen praxisrelevanten Anwendungsfällen ein umfangreiches Datenvolumen entsteht und die Replikation der Daten zwischen den Knoten in Folge dessen oft nicht mehr sinnvoll möglich ist, wird der Einsatz eines verteilten Dateisystems auf einem Shared Storage System notwendig. Die hierbei vorgestellten Prinzipien können ebenso auf gleichartige Dienste, wie z. B. einen hochverfügbaren File- oder Datenbankserver, übertragen werden.

Um die Hochverfügbarkeitslösungen schließlich abzurunden, veranschaulicht das letzte Konfigurationsbeispiel einen Datenbankserver, welcher als **MySQL-Cluster** betrieben wird. Behandelt werden die Grundlagen zum Betrieb eines hochverfügbaren MySQL-Servers sowie verschiedene Konfigurations- und Betriebsarten ausführlich beleuchtet. Da der MySQL-Cluster keine integrierte Möglichkeit zur dynamischen Lastverteilung besitzt, werden unterschiedliche Verfahren hierfür analysiert und für einen Software Load Balancer entsprechende Beispielkonfiguration angegeben. Um die Skalierbarkeit des MySQL-Clusters besser beurteilen zu können, befinden sich im letzten Teil Benchmarkergebnisse unterschiedlichster Konfigurationsarten für eine abschließende Bewertung.

4.2 FTP-HA-Cluster mit DRBD

4.2.1 Allgemeines

Dieses erste einfache Konfigurationsbeispiel demonstriert einen FTP-HA-Cluster bestehend aus zwei Knoten, die sich gegenseitig mit Heartbeat überwachen. Da kein Shared Storage zur Verfügung steht, erfolgt die Synchronisation der Daten mit dem Distributed Replicated Block Device (DRBD). Es dient zur Vorführung der grundlegenden Funktion eines Failover-Clusters mit Linux-HA Version 1 und wird sinnvollerweise durch DRBD für eine kostengünstige Clusterlösung ergänzt.

Jeder Knoten besitzt eine interne IP-Adresse für den Nachrichtenaustausch über Heartbeat und zur Administration. Zusätzlich dazu erhält der aktive Knoten eine virtuelle IP-Adresse auf demselben Netzwerkinterface zur Kommunikation mit den Clients. Bei Ausbleiben des Heartbeats des aktiven Knotens, übernimmt der passive alle Ressourcen. Das betrifft im konkreten Fall die VIP, die konfigurierte DRBD-Ressource (Änderung des Sekundär- in den Primärstatus), Einhängen des Dateisystems sowie den Start des FTP-Servers.

Dieses Beispiel umfasst die Grundkonfiguration von Heartbeat und stellt die Einrichtung von DRBD ausführlich dar. Auf Basis dieser Konfigurationsansätze kann eine Vielzahl von Diensten, welche einen gemeinsamen Datenbestand benötigen, als Failover-Cluster hochverfügbar gemacht werden.

4.2.2 Installation

Die Beispielkonfiguration setzt sich aus den folgenden Knoten zusammen:

- FTP-Server ha-ftp1 (IP-Adresse: 192.168.0.1)
- FTP-Server ha-ftp2 (IP-Adresse: 192.168.0.2)

Zur Kommunikation mit den Clients dient die durch Heartbeat verwaltete VIP 192.168.0.5.

Software

Die nachfolgend aufgeführte Software wird auf den beiden Knoten benötigt:

- Heartbeat zur Überwachung der Knoten
- DRBD zur Replikation des Datenbestandes
- IP-Route zur Verwaltung des Netzwerksubsystems durch Heartbeat
- FTP-Server (z. B. ProFTPD, vsftpd, Pure-FTPd)

Auf konkrete Paketangaben und nachfolgend auch Konfigurationsdetails wird verzichtet, um distributionsunabhängig zu bleiben.

4.2.3 Konfiguration und Test von DRBD

[LINe] [Sch09]

Nachdem das Kernelmodul erfolgreich kompiliert und die Userspace-Tools von DRBD installiert wurden, ist das Anlegen der Konfigurationsdatei `drbd.conf` erforderlich, die auf beiden Knoten identisch sein muss. Ein Beispiel einer Minimalkonfiguration, findet man im Anhang A.2 auf Seite 92. In dieser wird die Ressource "pg" definiert, welche jeweils die lokalen Blockgeräte `/dev/sda6` als virtuelles Blockgerät `/dev/drbd0` verwendet. Für die Erläuterung sowie weiteren Konfigurationsoptionen sei an dieser Stelle auf die ausführlich kommentierte Original-Konfigurationsdatei `drbd.conf` oder die Dokumentation von DRBD [LINe] verwiesen. Für den Einsatz im Cluster sollte der Administrator dennoch ein besonderes Augenmerk auf die drei Optionen *after-sb-** legen, falls dieser in eine Split Brain-Situation gerät und die Knoten sich gegenseitig nicht mehr erreichen. Diese Optionen konfigurieren das Verhalten von DRBD, sobald das Kommunikationsproblem behoben ist. Das trifft natürlich nur dann zu, sofern beide Knoten in dieser Zwischenzeit auf den Primärzustand gewechselt haben und nicht mehr klar ist, welchen Daten autoritativ sind.

Für die Initialisierung des Blockgerätes mit den Metadaten der Ressource "pg" für DRBD, muss auf beiden Knoten folgender Befehl ausgeführt werden:

```
# drbdadm create-md pg
```

Sofern die Metadaten auf dem gleichen Blockgerät gespeichert werden, ist eine Partitionsgröße von mindestens 256 MB erforderlich. Bei kleineren Partitionen erhält man einen entsprechenden Hinweis durch `drbdadm`. Ferner können die Metadaten auf einem anderen Blockgerät gespeichert werden, was auch zu einer besseren Performance führt. Die Größe der Metadaten kann anhand der folgenden Formel exakt bestimmt werden:

$$M_S = \frac{C_S}{2^{18}} * 8 + 72$$

C_S ist die Größe des Laufwerks in Sektoren, welche man mit folgendem Befehl ermittelt, der auch für Laufwerksgrößen mit mehr als 2 TB funktioniert:

```
# echo $(( $(blockdev --getsize64 device) / 512 ))
```

Damit die entsprechenden DRBD-Geräte für die Ressource "pg" im Dateisystem angelegt und diese mit den konfigurierten logischen HDD-Partitionen verbunden werden, führt man dazu folgendes Kommando auf beiden Knoten aus:

```
# drbdadm attach pg
```

Weiterhin muss auf beiden Knoten die Synchronisierung der DRBD für die Ressource “pg” mit dem Befehl

```
# drbdadm syncer pg
```

konfiguriert werden.

Um nun die Ressourcen letztendlich miteinander zu verbinden, muss das folgende Kommando auf beiden Knoten möglichst zeitnah ausgeführt werden, bevor das konfigurierte Zeitlimit für den Zusammenschluss der Ressourcen überschritten wird:

```
# drbdadm connect pg
```

Alternativ werden die drei genannten Befehle (attach, syncer, connect) mit dem Befehl

```
# drbdadm up
```

hintereinander ausgeführt.

Den Status der Ressourcen findet man im Dateisystem unter `/proc/drbd`.

```
# cat /proc/drbd
version: 8.3.3 (api:88/proto:86-91)
GIT-hash: 49bfeeaf3690ad0b9afd5376feda9e9eb34a30f3 build by pg
        0: cs:Connected ro:Secondary/Secondary ds:Inconsistent/Inconsistent C r----
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:1052152
```

Listing 4.1: Status der DRBD-Ressourcen nach dem Verbinden

Wie im Listing 4.1 dargestellt, erkennt man, dass beide Knoten miteinander verbunden sind, jedoch die jeweilige Partition als Secondary eingebunden ist und der Datenbestand inkonsistent ist. Das ist insofern korrekt, da bisher nicht festgelegt wurde, welcher Knoten den Primärstatus übernehmen soll und keinerlei sinnvolle Daten geschrieben wurden.

Im vorliegenden Beispiel wird mit dem Kommando

```
# drbdadm -- --overwrite-data-of-peer primary pg
```

die Primärpartition für den Host ha-ftp1 eingerichtet und die erste vollständige Synchronisation ausgeführt, indem die Daten auf dem sekundären Host ha-ftp2 überschrieben werden. Falls auf einer Partition bereits DRBD eingerichtet ist, sollte diese selbstverständlich mit dem Primärstatus versehen werden.

Der Fortschritt der Synchronisation zwischen den Knoten kann im Status der DRBD-Ressourcen verfolgt werden.

```
ha-ftp1:~# cat /proc/drbd
version: 8.3.3 (api:88/proto:86-91)
GIT-hash: 49bfeeaf3690ad0b9afd5376feda9e9eb34a30f3 build by pg@ha-ftp1
 0: cs:SyncSource ro:Primary/Secondary ds:UpToDate/Inconsistent C r----
    ns:258048 nr:0 dw:0 dr:258252 al:0 bm:15 lo:0 pe:16 ua:0 ap:0 ep:1 wo:b oos:794616
        [====>..............] sync'ed: 25.0% (794616/1052152)K
        finish: 0:01:16 speed: 10,336 (10,300) K/sec
```

Listing 4.2: Status der DRBD-Ressourcen beim Synchronisieren

In der Zwischenzeit ist die Nutzung der Partition bereits möglich. Es ist jedoch empfehlenswert die Synchronisation erst vollständig durchführen zu lassen, bevor ein Dateisystem und Daten darauf angelegt werden. Nach erfolgreichem Abschluss der Synchronisation sollte der Status auf dem Primärknoten wie folgt aussehen:

```
ha-ftp1:~# cat /proc/drbd
version: 8.3.3 (api:88/proto:86-91)
GIT-hash: 49bfeeaf3690ad0b9afd5376feda9e9eb34a30f3 build by pg@ha-ftp1
 0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r----
    ns:1052152 nr:0 dw:0 dr:1052356 al:0 bm:65 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

Listing 4.3: Status der DRBD-Ressourcen nach erfolgreicher Synchronisation

Des Weiteren kann nun ein lokales Dateisystem (z. B. ext3, XFS) auf der primären Partition `/dev/drbd0` angelegt werden. Es sollte hinreichend getestet werden, ob DRBD korrekt arbeitet, indem die Synchronisation der Daten überprüft wird.

Durch das Setzen des Sekundärstatus der Ressource “pg” auf ha-ftp1 mit dem Befehl

```
ha-ftp1:~# drbdadm secondary pg
```

sowie die Übernahme des Primärstatus dieser auf ha-ftp2

```
ha-ftp2:~# drbdadm primary pg
```

kann die korrekte Synchronisation der Daten verifiziert werden, sofern auf dem Host ha-ftp2 derselbe Inhalt wie zuvor auf ha-ftp1 beim Einhängen des Dateisystems erscheint.

Die Kommunikation bei DRBD erfolgt nicht auf redundantem Weg, so dass der Einsatz eines virtuellen Netzwerkinterfaces im Trunk von Vorteil ist. Das sichert einerseits die clusterinterne Kommunikation redundant ab und verbessert andererseits die Performance der Synchronisation bei größeren Blockgeräten erheblich.

4.2.4 Konfiguration von Heartbeat

Die Konfiguration von Heartbeat erfolgt anhand der drei Konfigurationsdateien `ha.cf`, `haresources` sowie `authkeys`, die auf beiden Knoten den gleichen Inhalt haben müssen. Im Anhang A.2 auf Seite 92 befinden sich dafür Konfigurationsbeispiele. Alternativ können die mitgelieferten Konfigurationsdateien der Heartbeat-Dokumentation (`doc`-Verzeichnis) genutzt werden. Des Weiteren sind die wichtigsten Konfigurationsoptionen im Anhang A.4 auf Seite 96 erläutert, so dass auf Details verzichtet wird.

Die Datei `authkeys` muss die Dateirechte 0600 aufweisen, da sonst der Heartbeat-Dienst den Start verweigert. Ein Blick ins Logfile bringt hier jedoch schnell Aufschluss. Weiterhin müssen alle Mitglieder im Cluster den korrekten Hostname besitzen, welcher in der Datei `ha.cf` konfiguriert ist und mit dem Befehl

```
# uname -n
```

angezeigt werden kann. Sind beide Knotennamen nicht identisch, verweigert Heartbeat auch hier den Start.

Die Ressourcen werden nach dem folgenden Schema in der Datei `haresources` im Linux-HA Version 1 Stil wie folgt definiert:

```
node-name resource1 resource2 ... resourceN
```

Anhand des Konfigurationsbeispiel im Anhang sollen die genutzten Scripte und deren Funktion kurz erläutert werden:

- IPaddr2::192.168.0.5/24/eth0/192.168.0.255
  Konfiguriert die VIP 192.168.0.5/24 für das Netzwerkinterface eth0
- drbddisk::pg
  Verwaltung der DRBD-Ressource “pg”
- Filesystem::/dev/drbd0::/shared-ftp::xfs
  Einhängen des XFS-Dateisystems unter `/shared-ftp`
- proftpd
  Starten des ProFTPD-Servers durch Init-Script

Nachdem alle Konfigurationsdateien angelegt wurden, sollten diese auf beiden Knoten identisch vorliegen, damit Heartbeat mit dem Befehl

```
# /etc/init.d/heartbeat start
```

gestartet werden kann. Beim Start ist auf Fehlermeldungen in der Konsole sowie im Logfile zu achten, damit eine einwandfreie Funktion gewährleistet werden kann.

4.3 Webserver-HA-Cluster mit Load Balancer

4.3.1 Allgemeines

Diese Beispielkonfiguration beschreibt die Kombination von Linux-HA Version 1 mit dem Linux Virtual Server (LVS), um damit einen hochverfügbaren sowie hoch skalierbaren Webcluster umzusetzen. Basierend auf dem Prinzip wie in Abbildung 3.4 auf Seite 53 dargestellt, erhalten die Webserver einen Load Balancer (LB), welcher die Anfragen entgegennimmt und an die verfügbaren Webserver weiterleitet. Durch eine Erweiterung mit `ldirectord` können die Webserver überwacht und auf Basis dieser Statusinformationen die Weiterleitungstabelle des LVS dynamisch gepflegt werden. Die Kommunikation mit dem Webcluster erfolgt durch die Verwendung einer virtuellen IP-Adresse (VIP) für die Applikationen völlig transparent.

Um einen Single Point of Failure (SPOF) zu vermeiden, wird der LB redundant ausgelegt. Der Heartbeat-Dienst in Linux-HA übernimmt hierbei die Überwachung der beiden LB. Fällt der aktive aus, werden die Ressourcen auf den passiven Knoten umgeschwenkt. Das betrifft im Fehlerfall die VIP des Clusters, den Dienst für den Abgleich der Verbindungstabellen des LVS sowie den `ldirectord`-Prozess zur Überwachung der Webserver und Verwaltung des Pools.

Damit alle Webserver auf den gleichen Datenbestand zugreifen können, wird ein verteiltes Dateisystem verwendet. In diesem Beispielszenario kommt das Oracle Cluster File System 2 (OCFS) zum Einsatz, da es mittlerweile ausgereift und schon seit längerem fester Bestandteil des Kernels ist und sich dazu auch noch einfach konfigurieren lässt. Es entfallen dadurch aufwändige Kernel Patches, wie z. B. bei Lustre, die nur für bestimmte Kernel-Versionen freigegeben sind. Weiterhin ist für den Erhalt einer Session im Fehlerfall eines Webservers die Nutzung eines gemeinsamen Datenbestandes notwendig, da alle Webserver die Session-ID in Form einer Textdatei serverseitig ablegen. Die Übertragung einer Session-ID erfolgt bei HTTP durch Cookies, Einfügen in die Uniform Resource Identifiers (URI) oder unsichtbare Formularfelder.

4.3.2 Installation

Die Demokonfiguration besteht aus den folgenden vier Knoten:

- Load Balancer ha-lb1 (IP-Adresse: 192.168.1.1)
- Load Balancer ha-lb2 (IP-Adresse: 192.168.1.2)
- Webserver ha-web1 (IP-Adresse: 192.168.1.3)
- Webserver ha-web2 (IP-Adresse: 192.168.1.4)

Zur Kommunikation mit dem Webcluster wird die VIP-Adresse 192.168.1.5 verwendet.

Software

Die folgende Software wird auf den Load Balancern benötigt:

- Heartbeat zur Überwachung der Knoten
- IP-Route zur Verwaltung des Netzwerksubsystems durch Heartbeat
- LVS - IPVSADM zur Administration des IP Virtual Servers im Kernel
- Ultra Money - Linux Director Daemon (`ldirectord`)

Weiterhin muss die nachstehende Software auf den Webservern installiert werden:

- Apache-Webserver (optional: mit PHP und MySQL)
- Oracle Cluster File System 2 Tools (optional: GUI)

Auf konkrete Paketangaben und nachfolgend auch Konfigurationsdetails wird auch in diesem Beispielszenario verzichtet, um weiterhin distributionsunabhängig zu bleiben.

4.3.3 Konfiguration

Für alle Konfigurationsdateien werden Beispiele im Anhang A.3 auf Seite 94 mitgeliefert.

Konfiguration von Heartbeat

Die Einrichtung von Heartbeat erfolgt analog wie in Kapitel 4.2.4 auf Seite 68 beschrieben. Es unterscheidet sich im Wesentlichen lediglich die Konfiguration der Ressourcen, die durch Heartbeat verwaltet und hier kurz beschrieben werden sollen:

- IPaddr2::192.168.1.5/24/eth0/192.168.1.255
  Konfiguriert die VIP 192.168.1.5/24 für das Netzwerkinterface eth0
- LVSSyncDaemonSwap::master
  Start des Linux Virtual Server State Sync Daemon als Master
- ldirectord::ldirectord.cf
  Start des Linux Director Daemons mit Konfigurationsdatei `ldirectord.cf`

Konfiguration von LVS - ldirectord

Für den Betrieb des Linux Director Daemons ist die Erstellung der Konfigurationsdatei `ldirectord.cf` notwendig, die auf beiden Load Balacern identisch sein muss. Darin werden alle benötigten Parameter erfasst, die das Kernelmodul des LVS für die korrekte Funktionsweise benötigt. Beim Anlegen der Datei ist besonders auf die Tabulatoren zu achten, die in den Beispielen entsprechend mit einer Linie gekennzeichnet wurden. Eine ausführliche Übersicht ausgewählter Optionen findet man im Anhang A.5 auf Seite 99. Für weitere Konfigurationsoptionen sei ausdrücklich auf die Manpage verwiesen.

Falls direktes Routing, wie im Konfigurationsbeispiel, oder IP-Tunneling zur technischen Paketweiterleitung genutzt wird, muss die VIP des Clusters auf dem Loopback-Device (lo) der Webserver statisch eingerichtet werden, damit die Pakete auch den jeweiligen Server erreichen. Außerdem kann auf diese Weise am einfachsten verhindert werden, dass die Webserver auf ARP-Anfragen antworten, indem die folgenden Kernelparameter (Deaktivierung des ARP-Announcements und ARP-Replys) in der Datei `sysctl.conf` gesetzt werden

```
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 2
net.ipv4.conf.eth0.arp_ignore = 1
net.ipv4.conf.eth0.arp_announce = 2
```

Listing 4.4: Kernelparameter zum ARP-Verhalten

und durch das Kommando

```
# sysctl -p
```

im laufenden Betrieb übernommen werden können, ohne einen Neustart des Systems durchzuführen. Falls diese Parameter nicht gesetzt werden, führt dies zu Funktionsstörungen beim Load Balancer, da dieser und die Real Server auf ARP-Anfragen für die VIP des Clusters eines Routers bzw. Clients gleichzeitig antworten würden.

Konfiguration des Oracle Cluster File Systems 2

Für das Anlegen der Konfiguration des OCFS2 stehen zwei Möglichkeiten zur Verfügung - zum einen das grafische Werkzeug `ocfs2console` oder die manuelle Erstellung der Datei `cluster.conf`, die ebenso auf allen Knoten identisch vorliegen muss. Auch dabei ist wieder besonders auf die Tabulatoren zu achten, die entsprechend in den Beispielen gekennzeichnet wurden, da sonst der Dienst für das verteilte Dateisystem nicht gestartet werden kann. Je nach Anzahl der Mitglieder im Cluster, muss für jeden Knoten die Option *number* um eins inkrementiert werden. In der Sektion *cluster* ist die Gesamtanzahl der Knoten (*node_count*) anzugeben.

Für die Inbetriebnahme von OCFS2 muss das Dateisystem von einem beliebigen Knoten aus mit dem Kommando

```
# mkfs.ocfs2 /dev/sdb
```

formatiert werden. Anschließend kann der Clusterstack mit dem Befehl

```
# /etc/init.d/o2cb start
```

auf dem jeweiligen Knoten gestartet und das Dateisystem von den konfigurierten Knoten eingehangen werden. Bei Bedarf können bestimmte Parameter des Dateisystems nachträglich mit `tunefs.ocfs2` geändert werden.

Konfiguration des Apache-Webservers mit PHP

Damit der Webserver die Daten aus dem verteilten Dateisystem ausliefert, muss die Option *DocumentRoot* in der Datei `apache2.conf` entsprechend angepasst werden. Im Übrigen muss die PHP-Konfiguration mithilfe der Option *session.save_path* in der Datei `php.ini` so geändert werden, dass die Session-Dateien ebenso im gemeinsamen Datenbereich abgelegt werden und dadurch alle Webserver auf die gleichen Sessions zugreifen können, was für den Erhalt einer Session im Failover-Fall wichtig ist.

4.3.4 Verifizierung der Konfiguration

Um die korrekte Funktionsweise der einzelnen Bestandteile der Konfiguration zu prüfen, sollten die nachfolgenden Schritte durchgeführt werden:

Netzwerkkonnektivität

Die Ausgabe zeigt alle konfigurierten IP-Adressen des Netzwerkinterfaces eth0. Nach dem Start von Heartbeat, sollte auf dem Primärknoten des LB die VIP des Clusters als weitere Adresse eingerichtet sein.

```
ha-lb1:~# ip addr sh eth0
```

```
eth0:<BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:50:56:8e:40:82 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global eth0
    inet 192.168.1.5/24 brd 192.168.1.255 scope global secondary eth0
    inet6 fe80::250:56ff:fe8e:4082/64 scope link
       valid_lft forever preferred_lft forever
```

Listing 4.5: Webserver-HA-Cluster - Netzwerkprüfung

Prozessstatus Linux Director Daemon

Damit die Überprüfung der Webserver erfolgt, sollte auch der Linux Director Daemon auf dem aktiven Knoten gestartet sein, auf dem passiven jedoch gestoppt sein.

```
ha-lb1:~# ldirectord ldirectord.cf status
```

```
ldirectord for /etc/ha.d/ldirectord.cf is running with pid: 11824
```

Listing 4.6: Webserver-HA-Cluster - ldirectord Master-Prozessstatus

Prozessstatus LVS State Sync Daemon

Zum Abgleich der Verbindungstabellen des LVS, die bei einem Failover wichtig sind, ist die korrekte Arbeitsweise im entsprechenden Status zu prüfen und auf dem aktiven LB folgende Ausgabe liefern:

```
# $PATH$/LVSSyncDaemonSwap master status
```

```
master running
(ipvs_syncmaster pid: 11763)
```

Listing 4.7: Webserver-HA-Cluster - LVS State Sync Daemon Master-Prozessstatus

Auf dem sekundären LB sollte eine ähnliche Ausgabe, jedoch mit dem Parameter `backup` anstelle von `master`, erfolgen.

LVS-Weiterleitungstabelle

In dieser Übersicht sind alle konfigurierten Real Server dargestellt. Diese gibt Auskunft über den Scheduling-Algorithmus für die VIP des Clusters, den Mechanismus zur technischen Paketweiterleitung für die jeweiligen Server, deren Gewichtung, aktive sowie inaktive Verbindungen.

```
# ipvsadm -L -n
```

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.5:80 rr
  -> 192.168.1.4:80               Route   1      0          0
  -> 192.168.1.3:80               Route   1      0          0
```

Listing 4.8: Webserver-HA-Cluster - LVS-Weiterleitungstabelle

Sobald ein konfigurierter Server nicht in der Liste erscheint oder dessen Gewichtung gleich 0 ist, werden keine Verbindungen mehr zu diesem weitergeleitet. Ein häufiger Fehler ist das Nichtbeachten der Groß- und Kleinschreibung, etc. der zu prüfenden Zeichenkette, was zu einer scheinbaren Nichterreichbarkeit eines Servers führt.

LVS-Verbindungstabelle

Diese Übersicht der aktuellen Clientverbindungen ist rein informativ, kann jedoch bei der Suche nach Fehlern hilfreich sein.

```
# ipvsadm -L -n -c
```

```
IPVS connection entries
pro expire state       source             virtual            destination
TCP 14:57  ESTABLISHED 192.168.1.10:35157 192.168.1.5:80     192.168.1.3:80
TCP 01:10  FIN_WAIT    192.168.1.10:35156 192.168.1.5:80     192.168.1.4:80
```

Listing 4.9: Webserver-HA-Cluster - LVS-Verbindungstabelle

Manuelles Failover mit Heartbeat

Zum Test oder zur externen Überwachung stehen die beiden Scripte `hb_standby` und `hb_takeover` in Heartbeat zur Verfügung, mit denen Ressourcen manuell zwischen den Knoten bewegt werden können. Beide Scripte verfügen über die Paramater `local`, `foreign` und `all`. Bei Ausführung des Scripts `hb_takeover` werden je nach übergebenem Parameter die Ressourcen vom jeweils anderen Knoten übernommen. Das Gegenteil bewirkt das Script `hb_standby`, welches die jeweiligen Ressourcen auf dem Knoten beendet, damit der andere diese übernehmen kann.

4.3.5 Auftretende Probleme

Sobald die Webserver nicht auf den gleichen Datenbestand zugreifen und die Weiterleitung nicht immer zum selben Real Server erfolgt, geht dabei die Session verloren. LVS bietet zur Lösung des Problems sog. Persistent Connections. Dabei werden die Verbindungen vom selben Client oder Netzwerk für eine konfigurierbare Zeit immer wieder zum gleichen Real Server weitergeleitet. Das hat letztendlich zur Folge, dass die Lastverteilung nicht mehr pro Verbindung, sondern pro Client erfolgt. Außerdem kann die Verwendung von Proxy-Farmen oder Cache-Clustern auf der Clientseite zum Verlust der Zuordnung führen. In diesem Fall muss das ganze Subnetz dafür konfiguriert werden (Granularity). Weiterhin problematisch ist der Ausfall eines Real Servers, sofern dieser nicht komplett aus der Serverliste gelöscht wird, sondern nur die Gewichtung auf 0 gesetzt wird. Die meisten Programme zur dynamischen Verwaltung der Verbindungstabellen arbeiten nach diesem Prinzip. Der Director leitet die Verbindungen trotz Ausfalls dann weiterhin auf diesen weiter. In beiden Fällen kommt es jedoch zum Verlust der Session-Information. Daher sollte ein Shared Storage oder ein Layer 7 LB eingesetzt werden, sofern der Verlust der Session für die individuelle Anforderung nicht tragbar ist. Aufgrund der genannten Probleme mit persistenten Verbindungen ist die Option *quiescent* des `ldirectord` besonders hervorzuheben. Sobald diese auf “no” gesetzt ist und ein Real Server ausfällt, wird dieser komplett aus der Weiterleitungstabelle gelöscht, wohingegen bei “yes” die Gewichtung auf 0 gesetzt wird.

Damit eine Session auch beim Übergang von HTTP zu HTTPS gerettet werden kann, steht eine sog. Catch-All Persistence zur Verfügung, die LVS auf Port 0 mit beständigen Verbindungen definiert. Daraufhin verwaltet LVS die Clientanfragen in einer eigenen Zuordnungstabelle unabhängig von den Verbindungsports. Damit bleibt die Session auch beim Wechsel von Port 80 (HTTP) auf Port 443 (SSL) erhalten, da der Port 0 einfach als Catch-All für alle Ports steht. Ein großer sicherheitskritischer Nachteil ist damit die Öffnung des kompletten Directors, da dieser ab sofort Verbindungen auf allen Ports entgegennimmt und auf die Real Server verteilt. Sofern diese Konfiguration erforderlich ist, sollten auf den Real Servern entsprechende Paketfilterregeln erstellt werden. Falls diese Möglichkeit nicht in Frage kommt, besteht eine Erweiterung für LVS, die mit Firewall Marks arbeitet und dadurch mehrere lastverteilte Dienste zusammenfassen kann. Bei Bedarf findet man weiterführende Informationen dazu unter [Mac09].

4.3.6 Alternative Möglichkeiten zur Lastverteilung

DNS Load Balancing

Alternativ zu einem dedizierten Load Balancer besteht die Möglichkeit eine Verteilung auf verschiedene Server unter Verwendung des Domain Name Systems (DNS) umzusetzen. DNS erlaubt die Zuweisung mehrerer IP-Adressen zu einem Hostname. Man bezeichnet diese Gruppierung als Resource Record Set. Ein DNS Server liefert alle IP-Adressen aus, jedoch kann die Reihenfolge bei mehreren Abfragen hintereinander differieren. Der unter Linux bekannte DNS Server Berkeley Internet Name Domain (BIND) ermöglicht die Konfiguration der Reihenfolge in Form von zyklisch (Round Robin), zufällig oder fest.

Layer 7 Load Balancing

Als weitere Möglichkeit besteht die Lastverteilung auf Applikationsebene in Form eines Reverse Proxys (OSI Layer 7), was den Funktionsumfang des Directors erweitert. Dies ermöglicht z. B. die Trennung von statischem und dynamischem Inhalt auf unterschiedliche Webserver. Sofern diesen kein Shared Storage für die Sessionverwaltung zur Verfügung steht, kann man mit einem Load Balancer auf Applikationsebene auch dieses Problem beheben. Allerdings erfordern diese Art von LB eine deutlich höhere Rechenleistung, weil der Director die Pakete komplett lesen und das verwendete Protokoll analysieren muss. Die LVS-Entwickler verfolgten mit dem Kernel TCP Virtual Server (KTCPVS) [Zhab] genau diesen Ansatz, jedoch ist das zuletzt veröffentliche Release 0.0.18 vom Dezember 2004 nicht für den Produktiveinsatz zu empfehlen. Wer dennoch die Funktionalität eines Layer 7 LB benötigt, findet mit Pound [Aps], XLB [XLB] oder HAProxy [HAP] brauchbare Lösungen, die jedoch ausschließlich für HTTP- und HTTPS-Lastverteilung konzipiert wurden. Ergänzend sei noch zu erwähnen, dass auch hardwarebasierte Lösungen existieren.

4.4 MySQL-Cluster

[SM09b] [SM] [SM09d] [Bre07]

4.4.1 Grundlagen

MySQL-Cluster ist eine Technologie für das Clustering von Datenbanken in einer Shared-Nothing-Architektur. Dabei erfüllt jeder Knoten unabhängig und eigenständig seine ihm zugetragenen Aufgaben. Deshalb ist kein SPOF vorhanden, da jeder Knoten auf einem separaten Host betrieben wird. In Hinblick auf Virtualisierung ist dieser Ansatz eher veraltet, da hierbei die redundante Auslegung der jeweiligen Bestandteile, insbesondere der Management- und Storageknoten, des MySQL-Clusters eine Rolle spielt. MySQL-Cluster ist ein konventioneller MySQL-Server, der jedoch die geclusterte Speicherengine Network Data Base (NDB), anstelle von MyISAM, InnoDB oder ähnlichem, nutzt.

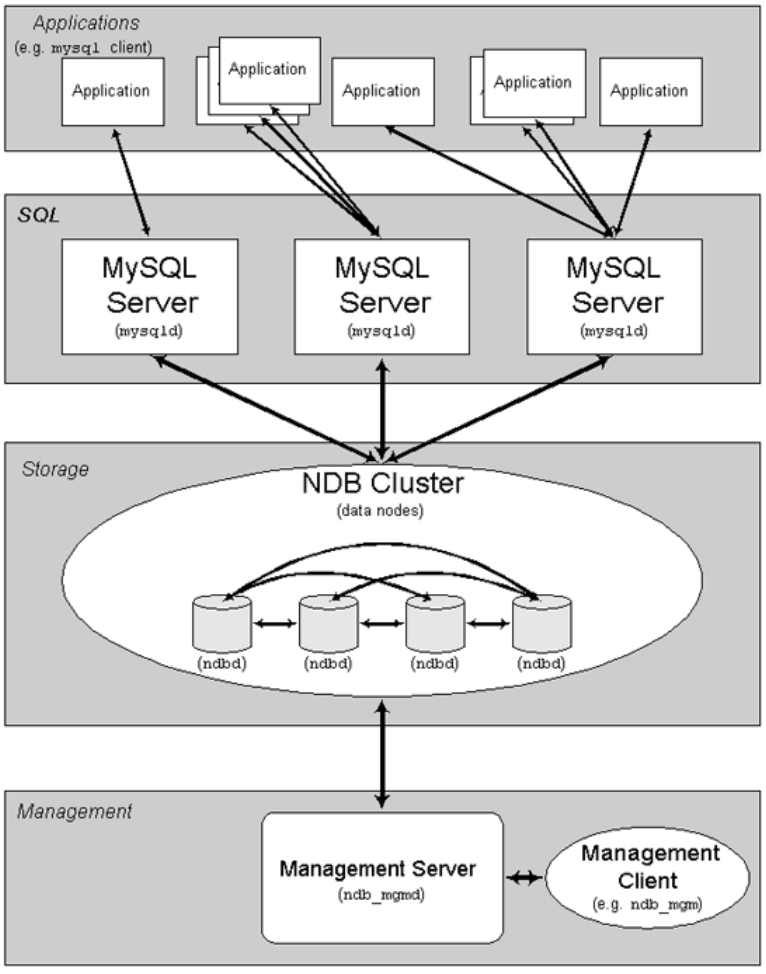

Abbildung 4.1: Aufbau eines MySQL-Clusters [SM09b]

Die in Abbildung 4.1 auf der vorherigen Seite dargestellten Komponenten bilden einen MySQL-Cluster, welcher ab Version 5 enthalten ist. Sobald die Speicherengine NDB genutzt wird, werden die Tabellen in den jeweiligen Datenknoten gespeichert, auf die alle MySQL-Server im Cluster direkt zugreifen können. Die Redundanz der Daten resultiert aus der Anzahl der Datenknoten sowie der Anzahl eingestellter Kopien. Die Performance (horizontale Skalierbarkeit) kann durch den Einsatz weiterer Knoten erheblich verbessert werden. Zusätzlich dazu erhöht sich die Verfügbarkeit bei entsprechender Konfiguration. Nach eigenen Angaben [SM] lässt sich damit eine Verfügbarkeit von 99,999% für geschäftskritische Anwendungen realisieren, was einer jährlichen Ausfallzeit unter sechs Minuten entspricht.

Die Minimalkonfiguration eines MySQL-Clusters besteht aus drei Knoten, je einem folgender Art:

- Management-Knoten (MGM-Knoten)
  Der MGM-Knoten verwaltet alle anderen Mitglieder im MySQL-Cluster und hält die komplette Einheit zusammen. Daher ist der Betrieb von mehreren MGM-Knoten in virtualisierten Umgebungen empfehlenswert, um eine bessere Verfügbarkeit zu erreichen, obwohl im Normalfall nur ein MGM-Node üblich ist, aber dennoch einen SPOF darstellt, sobald sich ein Knoten wieder am Cluster anmelden möchte. Über den MGM-Knoten können Kommandos an jeden Knoten im Cluster gesendet sowie deren Status abgefragt werden. Tritt eine Split Brain-Situation auf, übernimmt der MGM-Knoten die Entscheidung welcher Teilcluster weiterarbeiten soll.
- Datenknoten
  Dieser Knotentyp ist die Basis des Clusters, da er die Daten speichert. Jeder Node hält einen Teil der Gesamtdaten und entscheidet beim Eintreffen einer Anfrage, ob er diese beantworten kann oder weitere Datenknoten benötigt werden. Die Anzahl der benötigten Datenknoten im Cluster ergibt sich aus:

  $$\text{Anzahl Datenknoten} = \text{Anzahl der Kopien} * \text{Anzahl der Fragmente}$$

  Je nach Anzahl verfügbarer Prozessoren können mehrere `ndbd`-Prozesse innerhalb eines Systems gestartet werden.
- SQL-Knoten
  Die SQL-Anfragen der Applikation nehmen die SQL-Knoten entgegen. Es handelt sich hierbei um herkömmliche MySQL-Server, die mit der Storageengine NDB kommunizieren. Auch hier können je nach Anzahl verfügbarer CPUs mehrere `mysqld`- sowie `ndbd`-Prozesse auf einem Host gestartet werden.

Der Kernpunkt des MySQL-Clusters ist die Aufteilung der Daten. Die Partitionierung dieser übernimmt eine Funktion, die auf dem Primärschlüssel einer Tabelle basierend arbeitet - ist keiner vorhanden, wird ein versteckter Primärschlüssel eingefügt. Dabei werden die Tabellen in Fragmente zerlegt, die mehrfach auf verschiedene Datenknoten aufgeteilt werden. Anhand der eingestellten Anzahl von Kopien bilden sich Knotengruppen (Nodegroups). Alle Nodes innerhalb einer Gruppe enthalten dabei dieselben Tabellenfragmente. Deshalb ist es zwingend erforderlich, dass Datenknoten innerhalb einer Knotengruppe auf verschiedene physische Hosts verteilt werden, da sonst die eingestellte Redundanz nutzlos ist. Die Anzahl der Kopien ist auf vier begrenzt, was vierfache Redundanz bedeutet und selten sinnvoll ist. Üblich sind Konfigurationen mit zweifacher. Damit auf den komplette Datenbestand des Clusters zugegriffen werden kann, ist die korrekte Funktion eines Knoten pro Gruppe notwendig. Bei Ausfall eines Datenknotens erfolgt der Failover meist unter einer Sekunde, so dass die darunterliegende Applikation den Ausfall im Regelfall nicht bemerkt. Sobald der fehlerhafte Datenknoten wieder in den Cluster aufgenommen wird, erhält dieser die geänderten Fragmente völlig transparent und automatisch.

Jeder Node des MySQL-Clusters erhält eine ID (siehe Listing 4.13 auf Seite 82). Die Bildung der Nodegroups erfolgt implizit aus der Menge der Datenknoten mit den niedrigsten IDs. Das ist insofern wichtig, dass Knoten derselben Gruppe nicht auf derselben physischen Maschine betrieben werden. Die IDs werden beim ersten Start der einzelnen Knoten automatisch vergeben, können jedoch auch bei Bedarf manuell zugewiesen werden. Dies ist unter Umständen bei der Erweiterung von Datenknoten sinnvoll.

Der MySQL-Cluster arbeitet in der Standardkonfiguration mit arbeitsspeicherresidenten Tabellen (In-Memory Tables). Die Verwendung von In-Memory Tables ist jedoch nur beschränkt möglich und richtet sich vor allem nach dem Umfang der Datenbank. Zur Abschätzung der Arbeitsspeicherausstattung pro Datenknoten existiert folgende Formel:

$$\text{RAM} = \frac{\text{SizeofDatabase} \times \text{NumberOfReplicas} \times 1.1}{\text{NumberOfDataNodes}}$$

Da sämtliche Daten und Indizies im RAM untergebracht werden müssen, war der MySQL-Cluster für hochverfügbare Systeme bis Version 5.0 kaum zu gebrauchen. Mit Veröffentlichung von MySQL 5.1 wurden Disk Data Tables eingeführt. Damit können nicht indizierte oder mit Schlüsseln belegte Spalten auf der Festplatte abgelegt werden, was einem Großteil der gesamten Daten entspricht.

Für die Begriffsklärung von Knoten, Knotengruppen, Repliken und Partitionen ist das folgende MySQL-Cluster Dokument [SM09c] zu Rate zu ziehen. Diese Begriffe sind für das Verständnis zur Konfiguration und zum Betrieb eines MySQL-Clusters unerlässlich.

4.4.2 Installation

Für alle gängigen RPM-basierenden Linux-Distributionen stehen im Downloadbereich Pakete für den MySQL-Cluster zur Verfügung. Vorwiegend findet man die benötigte Software jedoch in der Paketverwaltung der jeweiligen Linux-Distribution.

Man benötigt folgende Pakete:

- Management-Knoten
  - Cluster storage engine management
  - (Cluster storage engine basic tools)
  - (Cluster storage engine extra tools)
- Datenknoten
  - Cluster storage engine
- SQL-Knoten
  - MySQL Server

Die in () gekennzeichneten Pakete sind optional. Weiterhin können je nach Bedarf die Pakete «Headers and libraries», «Shared libraries» sowie «Test suite» installiert werden. Sollte dennoch nicht die neuste Version zur Verfügung stehen, so kann das Komplettpaket von MySQL aus den Quellen übersetzt werden. Wichtig ist dabei, dass unbedingt die Option `--with-ndbcluster` mit angegeben wird - andere Konfigurationsparameter je nach Einsatzzweck.

4.4.3 Konfiguration

Für die Erstkonfiguration ist das HowTo [SM09d] der MySQL-Dokumentation sehr zu empfehlen. Folgende Konfigurationsdateien (Minimalkonfiguration ohne Performance-optimierung, etc.) sind auf dem jeweiligen Knotentyp anzulegen:

- Management-Knoten

```
# Options affecting ndbd processes on all data nodes:
[ndbd default]
NoOfReplicas=2   # Number of replicas
DataMemory=1024M # How much memory to allocate for data storage
IndexMemory=256M # How much memory to allocate for index storage

# Management process options:
[ndb_mgmd]
hostname=172.16.4.31            # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster  # Directory for MGM node log files

# Options for data node "A":
[ndbd]
hostname=172.16.4.33            # Hostname or IP address
datadir=/mysql/amdqc01          # Directory for this data node's data files

# Options for data node "B":
[ndbd]
hostname=172.16.4.34            # Hostname or IP address
datadir=/mysql/amdqc02          # Directory for this data node's data files

# SQL node options:
[mysqld]
hostname=172.16.4.32            # Hostname or IP address
```

Listing 4.10: MySQL-Cluster MGM-Node - `config.ini`

- SQL-Server

```
 # Options for mysqld process:
[mysqld]
ndbcluster                      # run NDB storage engine
ndb-connectstring=172.16.4.31  # location of management server
```

Listing 4.11: MySQL-Cluster SQL-Node - `my.cnf`

- Datenknoten

```
# Options for ndbd process:
[mysql_cluster]
ndb-connectstring=172.16.4.31  # location of management server
```

Listing 4.12: MySQL-Cluster NDB-Node - `my.cnf`

Die in der Konfiguration verwendeten Beispielwerte (IP-Adressen, Datenpfade, usw.) sind durch eigene zu ersetzen.

4.4.4 Inbetriebnahme

Zum Start des MySQL-Clusters muss jeder einzelne Prozess (`ndb_mgmd`, `ndbd`, `mysqld`) separat und auf dem jeweiligen Knoten gestartet werden. Sie können prinzipiell in beliebiger Reihenfolge gestartet werden, aber es ist dennoch empfehlenswert als erstes den Management-Knoten, dann die Speicherknoten und zum Schluss die SQL-Knoten zu starten:

- Management-Knoten:

  ```
  # ndb_mgmd -f $PATH$/config.ini
  ```

  Die Option `-f` oder `--config-file` gibt den Speicherort der MGM-Node Konfiguration an.

- Speicherknoten:

  ```
  # ndbd --initial
  ```

  Es ist äußerst wichtig, dass der Parameter `--initial` nur beim ersten Start der Speicherknoten, bei einem Neustart nach einer Sicherungs- / Wiederherstellungsoperation oder einer Konfigurationsänderung verwendet wird.
  Die Option `--initial` veranlasst den Knoten, alle für die Wiederherstellung erforderlichen Dateien zu löschen, die von vorherigen Speicherknoten angelegt wurden, einschließlich der Dateien des Redo-Logs.

- SQL-Knoten:

  ```
  # /etc/init.d/mysql start
  ```

Zur Überprüfung, ob alle Knoten korrekt hochgefahren wurden, sollte auf dem Management-Knoten der Befehl `# ndb_mgm` ausgeführt werden, und das Kommando `SHOW` die ähnlich folgende Ausgabe liefern:

```
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @172.16.4.33  (mysql-5.1.23 ndb-6.2.15, Nodegroup: 0, Master)
id=3    @172.16.4.34  (mysql-5.1.23 ndb-6.2.15, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @172.16.4.31  (mysql-5.1.23 ndb-6.2.15)

[mysqld(SQL)]   1 node(s)
id=4    @172.16.4.32  (mysql-5.1.23 ndb-6.2.15)
```

Listing 4.13: Status eines MySQL-Clusters

4.4.5 Bedienung

Anlegen von geclusterten Tabellen

Damit eine Tabelle geclustert wird, muss die Speicherengine NDB mit der Option `ENGINE=NDB` oder als Tabellenoption `ENGINE=NDBCLUSTER` angegeben werden:

```
CREATE TABLE tbl_name ( ... ) ENGINE=NDBCLUSTER;
```

Weiterhin kann eine bereits vorhandene Tabelle, die eine andere Speicherengine verwendet, wie folgt umgestellt werden:

```
ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
```

Jede NDB-Tabelle muss einen Primärschlüssel haben. Falls beim Anlegen der Tabelle kein Primärschlüssel definiert wird, generiert die Speicherengine NDB automatisch einen verborgenen Primärschlüssel, der selbstverständlich auch Platz wie jeder andere Tabellenindex belegt.

Sicheres Herunterfahren

Um den MySQL-Cluster herunterzufahren, gibt man den Befehl

```
# ndb_mgm -e shutdown
```

in eine Shell auf dem Host eines MGM-Knotens ein. Die Option `-e` übergibt einen Befehl von der Shell an den `ndb_mgm`-Client. Dadurch werden `ndb_mgm`-, `ndb_mgmd`- sowie alle laufenden `ndbd`-Prozesse beendet. Die jeweiligen SQL-Knoten können durch das Init-Script heruntergefahren werden.

Einschränkungen

Der MySQL-Cluster in Verbindung mit der Speicherengine NDB besitzt einige Einschränkungen gegenüber einem MySQL-Server mit lokaler Speicherengine. Es werden keine temporären Tabellen, keine Fulltext-Indizies sowie Fremdschlüssel unterstützt. Weiterhin müssen alle Mitglieder des Clusters von der gleichen Hardwarearchitektur sein. Pro Tabelle kann außerdem nur eine AUTO_INCREMENT-Spalte als Primärschlüssel verwendet werden. Eine vollständige Liste aller Einschränkungen findet man im MySQL-Referenzhandbuch unter [SM09a].

Disk Data Tables

Mit der Einführung von MySQL ab Version 5.1.6 ist es möglich, nicht indizierte Tabellen auf der Festplatte zu speichern. Bei früheren Versionen des MySQL-Clusters besteht nur die Möglichkeit, dass die kompletten Tabellen im RAM vorgehalten werden, was jedoch bei größeren Datenbanken nicht möglich ist.

Für die Erstellung von Disk Data Tables sind folgende Schritte notwendig:

1. Anlegen einer Logfile Group

```
CREATE LOGFILE GROUP lg_test
ADD UNDOFILE 'undo_1.dat'
INITIAL_SIZE 16M
UNDO_BUFFER_SIZE 2M
ENGINE NDB;
```

2. Anlegen eines Tablespace

```
CREATE TABLESPACE ts_test
    ADD DATAFILE 'data_1.dat'
    USE LOGFILE GROUP lg_test
    INITIAL_SIZE 32M
    ENGINE NDB;
```

3. Anlegen der Tabelle unter Angabe des bereits angelegten Tablespaces

```
CREATE TABLE test_1 (
    member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    last_name VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    joined DATE NOT NULL,
    INDEX(last_name, first_name)
    )
    TABLESPACE ts_test STORAGE DISK
    ENGINE NDB;
```

Eine Logfile Group sowie ein Tablespace kann je nach Einsatzzweck auch aus mehreren Dateien bestehen. Um weitere Dateien hinzuzufügen, verwendet man die folgenden Befehle:

```
ALTER LOGFILE GROUP lg_test
    ADD UNDOFILE 'undo_2.dat'
    INITIAL_SIZE 12M
    ENGINE NDB;
```

bzw.

```
ALTER TABLESPACE ts_test
    ADD DATAFILE 'data_2.dat'
    INITIAL_SIZE 48M
    ENGINE NDB;
```

4.4.6 Lastverteilung

Der MySQL-Cluster unterstützt keine Lastverteilung beim Einsatz mehrerer SQL-Knoten. Des Weiteren ist es von Vorteil, wenn Applikationen mit nur einer IP-Adresse kommunizieren und die Kommunikation im Backend völlig transparent erfolgt. Der in Kapitel 3.3.2 vorgestellte Linux Virtual Server (LVS) mit `ldirectord` zur Überwachung der SQL-Knoten ist dafür optimal geeignet. Der MySQL-Cluster kann damit eine virtuelle IP-Adresse für die Applikationen bereitstellen und die Lastverteilung erfolgt nach eingestelltem Scheduling-Algorithmus auf die vorhanden SQL-Knoten im Cluster. Die Überwachung erfolgt durch das regelmäßige Absenden einer Datenbankanfrage.

```
# Sample configuration for Linux Virtual Server - ldirectord.cf
#
# Global Directives
checktimeout=10
checkinterval=2
autoreload=no
logfile=local0
quiescent=yes

# Sample configuration for a MySQL virtual service
virtual = 192.168.1.1:3306          # Virtual IP:Port
        real=192.168.0.2:3306 gate  # IP SQL-Node 1:Port Direct Routing
        real=192.168.6.2:3306 gate  # IP SQL-Node 2:Port Direct Routing
        fallback=127.0.0.1:3306 gate # Fallback-IP
        checktype = negotiate
        login = "readuser"          # MySQL User
        passwd = "genericpassword"  # MySQL Password
        database = "portal"         # MySQL Database
        request = "SELECT * FROM link" # SQL-Query
        scheduler = wrr             # Scheduling-Algorithm
```

Listing 4.14: Beispielkonfiguration LVS für MySQL-Cluster - `ldirectord.cf`

Sofern LVS nicht genutzt werden soll, kann die Lastverteilung alternativ folgendermaßen erreicht werden:

- Round Robin DNS (siehe Kapitel 4.3.6)
- Linux HA und Round Robin DNS
  Als Erweiterung zum Round Robin DNS können die SQL-Knoten zusätzlich in Linux-HA eingebunden werden. Im Cluster Resource Manager (CRM) vergibt man dafür jedem Knoten eine VIP, die dann im DNS nach obigen Schema hinterlegt werden muss. Die Überwachung erfolgt über die üblichen Resource Agents oder eigene Scripts. Beim Ausfall eines Knotens migriert der CRM die VIP auf einen anderen SQL-Knoten, so dass alle im DNS eingetragenen IP-Adressen erreichbar bleiben.

- Hardware Load Balancer
  Ein Hardware Load Balancer ist deutlich professioneller und sicherer, jedoch auch eine sehr kostenintensive Angelegenheit. Die Funktionsweise ist ähnlich dem LVS. Es wird eine VIP für den Cluster sowie alle verfügbaren SQL-Knoten konfiguriert. Mithilfe von TCP-Portchecks oder Datenbankanfragen erfolgt die Überwachung. Weiterhin können bestimmte Werte über den Zustand der Datenbank abgefragt werden. Auf Basis dieser Daten kann eine noch bessere Lastverteilung im Gegensatz zum Software Load Balancer erfolgen. Da auch ein Hardware LB einen SPOF im Cluster darstellt, ist auch hier eine clusterfähige Lösung nötig, welche die Kosten und Komplexität weiter erhöht.
- JDBC Clustering
  JDBC bietet auf Applikationsebene eine Möglichkeit zum Clustering. Der Treiber unterstützt die Konfiguration mehrerer SQL-Server (Round Robin) und überwacht diese. Sofern alle eingesetzten Applikationen die JDBC-Schnittstelle nutzen, sollte diese Möglichkeit als Alternative in Betracht gezogen werden.

4.4.7 Alternative Konfigurationen

Falls der MySQL-Cluster aufgrund der genannten Einschränkungen nicht genutzt werden kann, sollte MySQL in Verbindung mit DRBD als Active-/Passive-Konfiguration betrieben werden. Es existieren kaum Active-/Active-Lösungen, bei der mehrere SQL-Server parallel auf den gleichen Datenbestand zugreifen. Die derzeit einzig sinnvolle verfügbare Lösung auf dem Markt hierfür ist Oracle RAC, welche mit enormen Lizenzkosten verbunden ist. Es ist kostengünstiger, diese in Hardware zu investieren.

Beim Betrieb von MySQL mit DRBD sollte Replikationsmodus C verwendet werden, damit die synchrone Datenübertragung zwischen beiden Servern sichergestellt werden kann. Im Fehlerfall schwenkt der CRM den Mountpoint des Datenverzeichnisses auf den anderen Knoten und startet dort den MySQL-Server. Auch hier ist die Verwendung einer gemeinsamen virtuellen IP-Adresse empfehlenswert, um möglichst transparent für Applikationen zu bleiben. Bei Verwendung von DRBD sollte die lokale Speicherengine InnoDB genutzt werden, weil die Datenbanktabellen beim Failover schnell, automatisiert und sicher wiederhergestellt werden können. Kommt es bei MyISAM-Tabellen zu Datenkorruptionen, müssen diese erst manuell repariert werden.

Als weitere Möglichkeit kommt die MySQL-Replikation [SM09e] in Frage, welche im Rahmen dieser Diplomarbeit nicht näher untersucht wurde.

4.4.8 Benchmarkergebnisse

Für die nachfolgenden Benchmarks wurden Testsysteme mit folgender Ausstattung verwendet:

- Motherboard Intel S3000PT / Intel S5000PA
- CPU Intel Core 2 Duo E6400 @ 2.13 GHz / Intel Xeon @ 2.66 GHz
- RAM 1 GB / 8 GB
- Festplatte Western Digital WD4000YS-01M
- NIC Intel Corporation 82573 Gigabit Ethernet Controller /
  Intel Corporation 80003ES2LAN Gigabit Ethernet Controller

Als Grundlage für die Benchmarks wurde SysBench [Kop] verwendet.

Der Benchmark wurde mit folgendem Aufruf auf dem jeweiligen Host gestartet:

```
sysbench --num-threads=8 --test=oltp --mysql-host=localhost --mysql-port=3306
--mysql-user=root --mysql-db=sysbench --mysql-table-engine=ndbcluster
--oltp-read-only=off --max-requests=0 --max-time=500
--oltp-test-mode=complex --oltp-table-size=1000000 run
```

Listing 4.15: SysBench: a system performance benchmark

Die Anzahl der gleichzeitig gestarteten Benchmarks richtet sich nach der Anzahl der eingesetzten SQL-Knoten. Sind in einer Konfiguration zwei MySQL-Server vorhanden, so wurde auf dem jeweiligen Host ein separater Benchmark gestartet, so dass die Datenknoten besser ausgelastet sind. Die Anzahl der Kopien wurde für alle Testfälle auf zwei gesetzt.

Config / Queries	1 mysqld, 4 ndbd
read	4.489.730
write	1.603.475
other	641.390
total	6.734.595
read per second	8.979
write per second	3.206

Tabelle 4.1: Benchmark: MySQL-Cluster 5.0 - In-Memory Table

Weitere Benchmarkergebnisse im Anhang A.6 auf Seite 101.

4.4.9 Schlussbemerkung

MySQL zählt mittlerweile zu den weit verbreitetsten relationalen Datenbanksystemen. Leider herrscht immer noch das Vorurteil, dass es sich bei MySQL um einen nicht transaktionssicheren "Zettelkasten" handelt. Dies stammt vor allem aus der Zeit von MySQL Version 3 - mittlerweile ist MySQL in Version 5 eine konkurrenzfähige Alternative. Sicherlich kann sich die OpenSource-Variante von MySQL nicht in allen Punkten mit einem Microsoft SQL-Server oder gar einer Oracle Database messen. Zudem ist kaum bekannt, dass sich mithilfe von MySQL hochverfügbare Datenbanken realisieren lassen und Clustering unterstützt wird.

Trotz allem ist der MySQL-Cluster keine Lösung für Performanceprobleme (vor allem Schreibzugriffe) jeglicher Art ohne entsprechende Hardwareausstattung, insbesondere des Storage-Systems. In einem MySQL-Cluster verursachen Schreibzugriffe einen großen Overhead. In weiteren Entwicklungsstufen ist eine Shared Storage-Lösung geplant, die die Schreibperformance gravierend verbessern soll. Es ist allerdings ein Trugschluss, dass mit zunehmender Knotenanzahl die verteilte Datenbank im Vergleich zu einem lokalen MySQL-Server immer bessere Antwortzeiten liefert. Jede Anfrage der Applikation wird zunächst an die SQL-Knoten gestellt, die diese wiederum an die Datenknoten weiterleiten. Falls der jeweilige Storagenode die Daten nicht vollständig liefern kann, müssen weitere Datenknoten befragt werden, was zu einer erhöhten clusterinternen Kommunikation führt. Aufgrund dessen sind die Mitglieder eines MySQL-Clusters mit 100-MBit- oder besser Gigabit-Ethernet zu betreiben. Außerdem spielen die Latenzzeiten im Ethernet eine wesentliche Rolle, so dass zum Erreichen der optimalen Performance der Einsatz serieller Hochgeschwindigkeits-Interconnects, wie z. B. InfiniBand oder SCI, erforderlich ist.

Bei einem lokalen MySQL-Server hingegen ist der beschränkende Faktor immer die HDD I/O-Performance. Bei Verwendung von Disk Data Tables im MySQL-Cluster kommt dieser Faktor zusätzlich zum Netzwerk hinzu. Sobald ein lokaler MySQL-Server nicht mehr genügend Performance liefert und auch bei Ausfall eines Servers die Verfügbarkeit nicht beeinträchtigt werden darf, sollte über den Einsatz eines MySQL-Clusters nachgedacht werden. Alternativ dazu ist die Variante der Active-/Passive-Konfiguration von MySQL in Verbindung mit DRBD in Betracht zu ziehen, falls die Verfügbarkeit des Datenbankservers von enormer Bedeutung ist und die Einschränkungen des MySQL-Clusters dessen Einsatz verhindern. Allerdings können die Wiederherstellungsvorgänge bei einem Failover zu einer kurzen Unterbrechung der Verfügbarkeit des Dienstes führen.

Kapitel 5

Fazit

5.1 Auswertung

Die derzeit entwickelten HA-Konfigurationen besitzen alle geforderten Eigenschaften, welche den Einsatz in hochverfügbaren virtualisierten Umgebungen unter den vorgegeben Rahmenbedingungen ermöglichen. Der ursprüngliche Ansatz M-to-N-Cluster mit den Funktionen der Virtualisierung zu kombinieren, musste jedoch verworfen werden. Die Komplexität des Systems wäre dabei exponentiell ansteigend. Da der Cluster Resource Manager keine Kenntnis davon hat, dass er sich in einer virtuellen Maschine befindet, ist es zum einen nicht möglich, genau zu definieren, welche Ressourcen auf welchem physischen Host vorrangig laufen sollen und zum anderen würden beide Technologien unter Umständen kontraproduktiv gegeneinander arbeiten.

Im Übrigen werden M-to-N-Cluster mit mehr als zwei Knoten von der ausgewählten Clusterlösung Linux-HA erst ab Version 2 unterstützt. Die Konfiguration erfolgt XML-basiert, das für den Linux-Systemadministrator eher ungewohnt ist. Aufgrund der mangelnden Abgrenzung der einzelnen Komponenten der Clusterlösung muss der Einsatz von Linux-HA ab Version 2 in einer hochverfügbaren produktiven Umgebung durchaus in Frage gestellt werden. Somit ergibt sich ein weiterer Grund, auf M-to-N-Cluster zu verzichten und anstelle davon auf herkömmliche Failover-Cluster zu setzen, die sich bereits mit Linux-HA Version 1 realisieren lassen. Weiterhin ist zweifache Redundanz für die meisten praktischen Anwendungsfälle ausreichend und auch ökonomisch sinnvoll. Dennoch kann mit einem Software Load Balancer eine hochverfügbare, hoch skalierbare Serverfarm im Sinne eines M-to-N-Clusters umgesetzt werden. Dazu müssen die Serverdienste auf den VMs innerhalb des Serverclusters so ausgelegt werden, dass auch beim Ausfall eines physischen Systems keine Beeinträchtigung der Gesamtperformance wahrgenommen wird. Der Vorteil der Virtualisierung ist hierbei eine effizientere Nutzung der Ressourcen.

5.2 Zusammenfassung

Als Ergebnis der Diplomarbeit lässt sich festhalten, dass Hochverfügbarkeit und Virtualisierung miteinander kombiniert werden können, sofern eine Farm von Servern betrieben wird, um die kritischen Dienste durch Platzierung der zugehörigen VMs auf verschiedene physische Hosts aufzuteilen. Da von Seiten der Virtualisierungssoftware oftmals nur der physische Server bzw. die jeweilige VM überwacht wird, ergänzt man die Verwaltung sowie das Monitoring der Serverdienste sinnvollerweise auf Applikationsebene mithilfe einer Cluster Suite und anderen Tools. Um das Potential beider Technologien bestmöglich zu nutzen, ist eine gründliche Planung Voraussetzung. Mit Einführung einer Virtualisierungslösung ergeben sich zudem eine Reihe von zusätzlichen nützlichen Funktionen, vor allem für den Einsatz in einer Clusterumgebung. Dazu zählt u. a. das flexiblere Management der Clusterknoten. Letztendlich ist die Auswahl der geeigneten Cluster- oder Virtualisierungslösung immer von der zu realisierenden Applikation und den Kundenanforderungen abhängig.

5.3 Ausblick

Neben der in dieser Diplomarbeit ausführlich dargestellten Clusterlösung Linux-HA, existiert alternativ die Red Hat Cluster Suite, die einen ähnlichen Funktionsumfang bietet und auf dem von Red Hat entwickelten Clusterkommunikationsstack OpenAIS basiert. Weiterhin ist der Betrieb dieser Lösung nur unter Red Hat Enterprise Linux möglich, so dass die geforderte Distributionsunabhängigkeit damit nicht eingehalten werden kann. Außerdem gibt es eine Reihe diverse proprietäre und eng an Hersteller gebundene Lösungen, wie z. B. SteelEyes LifeKeeper, die sich zu keiner Zeit nennenswert verbreitet haben. Daher sollte auch in Zukunft eher auf die Kombination von Pacemaker, Corosync und OpenAIS als distributionsunabhängige Lösung gesetzt werden. Es ist davon auszugehen, dass sich dieser Projektzusammenschluss in Kürze zu einer ausgereiften robusten Clusterlösung entwickelt und das Linux-HA Projekt in der ursprünglichen Form nicht mehr geben wird. Auch der Virtualisierungs-Trend hält weiterhin ungebrochen an. VMware bietet mittlerweile Desktop-Virtualisierung (VMware View 4) zur Bereitstellung von Desktops als Managed Service an. Insofern werden sich auch im Laufe der nächsten Jahre weiterhin neue Lösungen für den Einsatz in hochverfügbaren Umgebungen etablieren, welche möglicherweise den Einsatz einer Clusterlösung auf Applikationsebene überflüssig machen. Alternativ dazu existierten mittlerweile OCF-Agenten in Linux-HA zum Betrieb hochverfügbarer virtueller Maschinen, um beide Konzepte miteinander zu verschmelzen.

A Anhang

A.1 Downtime-Kalkulatoren

- http://www.dpair.com/DowntimeCalculator.aspx
- http://www.syan.co.uk/availability/DowntimeCalculator.aspx
- http://www.hwcs.com/software/sysb-ii/roi/calculator.asp
- http://users.telenet.be/phvg/AvailabilityTranslator.htm
- http://www.sudora.com/downtime.html
- http://www.nta-monitor.com/tools/downtime.html
- http://www.brixtec.com/DowntimeCalculator.php

A.2 FTP-HA-Cluster mit DRBD

```
# Sample configuration for Heartbeat - ha.cf
#
#       Facility to use for syslog()/logger
#
logfacility     local0
#
#       Set up a unicast / udp heartbeat medium
#
ucast eth0 192.168.0.1
ucast eth0 192.168.0.2
#       auto_failback "on" and "off" are backwards compatible with the old
#               "nice_failback on" setting.
#
auto_failback on
#       Tell what machines are in the cluster
#       node    nodename ...    -- must match uname -n
node    ha-ftp1
node    ha-ftp2
```

Listing A.1: FTP-HA-Cluster (Heartbeat) - `ha.cf`

```
# Sample configuration for Heartbeat - authkeys
#
# Authentication file.  Must be mode 600
#
auth 1
1 sha1 SecretKey
```

Listing A.2: FTP-HA-Cluster (Heartbeat) - `authkeys`

```
# Sample configuration for Heartbeat - haresources
#
#       This is a list of resources that move from machine to machine as
#       nodes go down and come up in the cluster.  Do not include
#       "administrative" or fixed IP addresses in this file.
#
# <VERY IMPORTANT NOTE>
#       The haresources files MUST BE IDENTICAL on all nodes of the cluster.
#
ha-ftp1 IPaddr2::192.168.0.5/24/eth0/192.168.0.255
ha-ftp1 drbddisk::pg
ha-ftp1 Filesystem::/dev/drbd0::/shared-ftp::xfs
ha-ftp1 proftpd
```

Listing A.3: FTP-HA-Cluster (Heartbeat) - `haresources`

```
# Sample configuration for DRBD - drbd.conf
#
global  {
        usage-count no;
        }

common {
        syncer { rate 100M; }
        }

resource pg {

        protocol C;

        handlers {
                pri-on-incon-degr "echo o > /proc/sysrq-trigger ; halt -f";
                pri-lost-after-sb "echo o > /proc/sysrq-trigger ; halt -f";
                local-io-error "echo o > /proc/sysrq-trigger ; halt -f";
        }

        startup {
                degr-wfc-timeout 120;
        }

        disk {
                on-io-error detach;
        }

        net {
                after-sb-0pri disconnect;
                after-sb-1pri consensus;
                after-sb-2pri disconnect;
                rr-conflict disconnect;
        }

        syncer {
                rate 10M;
                al-extents 257;
        }

        on ha-ftp1 {
                device    /dev/drbd0;
                disk      /dev/sda6;
                address   192.168.0.1:7788;
                meta-disk internal;
        }
        on ha-ftp2 {
                device    /dev/drbd0;
                disk      /dev/sda6;
                address   192.168.0.2:7788;
                meta-disk internal;
        }
}
```

Listing A.4: FTP-HA-Cluster (DRBD) - `drbd.conf`

A.3 Webserver-HA-Cluster mit Load Balancer

```
# Sample configuration for Heartbeat - ha.cf
#
#       Facility to use for syslog()/logger
#
logfacility     local0
#
#       Set up a unicast / udp heartbeat medium
#
ucast eth0 192.168.1.1
ucast eth0 192.168.1.2
#       auto_failback "on" and "off" are backwards compatible with the old
#               "nice_failback on" setting.
#
auto_failback on
#       Tell what machines are in the cluster
#       node    nodename ...    -- must match uname -n
node    ha-lb1
node    ha-lb2
```

Listing A.5: Webserver-HA-Cluster (Heartbeat) - `ha.cf`

```
# Sample configuration for Heartbeat - authkeys
#
# Authentication file.  Must be mode 600
#
auth 1
1 sha1 SecretKey
```

Listing A.6: Webserver-HA-Cluster (Heartbeat) - `authkeys`

```
# Sample configuration for Heartbeat - haresources
#
#       This is a list of resources that move from machine to machine as
#       nodes go down and come up in the cluster.  Do not include
#       "administrative" or fixed IP addresses in this file.
#
# <VERY IMPORTANT NOTE>
#       The haresources files MUST BE IDENTICAL on all nodes of the cluster.
#
ha-lb1  IPaddr2::192.168.1.5/24/eth0/192.168.1.255
ha-lb1  LVSSyncDaemonSwap::master
ha-lb1  ldirectord::ldirectord.cf
```

Listing A.7: Webserver-HA-Cluster (Heartbeat) - `haresources`

```
# Sample configuration for Linux Virtual Server - ldirectord.cf
#
# Global Directives
checktimeout=10
checkinterval=2
autoreload=no
logfile=local0
quiescent=yes

# Virtual Server for HTTP
virtual=192.168.1.5:80
        fallback=127.0.0.1:80
        real=192.168.1.3:80 gate
        real=192.168.1.4:80 gate
        service=http
        request="test.html"
        receive="Still alive"
        scheduler=rr
        #persistent=600
        protocol=tcp
        checktype=negotiate
```

Listing A.8: Webserver-HA-Cluster (LVS) - `ldirectord.cf`

```
# Sample configuration for Oracle Cluster File System 2 - cluster.conf
#
node:
        ip_port = 7777
        ip_address = 192.168.1.3
        number = 0
        name = ha-web1
        cluster = apache-ha
node:
        ip_port = 7777
        ip_address = 192.168.1.4
        number = 1
        name = ha-web2
        cluster = apache-ha
cluster:
        node_count = 2
        name = apache-ha
```

Listing A.9: Webserver-HA-Cluster (OCFS2) - `cluster.conf`

A.4 Konfigurationsoptionen - `ha.cf`

[Kle] [Sch09]

crm `on|off|respawn`

Vorgabewert: `off`

Diese Option aktiviert (`on`) bzw. deaktiviert (`off`) den Cluster Resource Manager. Um weiterhin Failover-Cluster im Linux-HA Version 1 Stil zu unterstützen, ist der Vorgabewert `off`. Sobald ein moderner Pacemaker-Cluster konfiguriert werden soll, muss dieser Wert explizit auf `on` gesetzt werden.

debugfile `Datei`

Vorgabewert: `/var/log/ha-debug`

Datei, in die Debug-Meldungen geschrieben werden. Für Linux-HA Version 2 ist diese Option deprecated - anstelle davon dient nun *use_logd*.

logfacility `Syslog-Einrichtung`

Vorgabewert: `local0`

Bestimmt das Logging-Verhalten von Heartbeat. Falls eine der Optionen *debugfile* oder *logfile* gesetzt ist, werden entsprechende Meldungen in die jeweils angegebene Datei geleitet. Standardmäßig werden mit dem Wert `local0` alle Meldungen an den `syslogd` weitergereicht. Für den Fall, dass keine der genannten Optionen gesetzt sind, wird der Vorgabewert von *debugfile* und *logfile* genutzt.

logfile `Datei`

Vorgabewert: `/var/log/ha-log`

Logdatei für alle anderen Meldungen. Für Linux-HA Version 2 ist diese Option ebenfalls deprecated - ersatzweise ist dafür nun *use_logd* zu verwenden.

keepalive `Intervall`

Vorgabewert: `2`

Gibt die Zeit zwischen zwei Heartbeat-Paketen an. Die Vorgabe für Zeiteinheiten bei Heartbeat ist Sekunden - andere Einheiten müssen exakt angegeben werden. Ein Wert von `500ms` entspricht somit einer halben Sekunde.

deadtime `Auszeit`

Vorgabewert: `30`

Die *deadtime* definiert die Ausfallzeit von Heartbeat-Paketen, bevor ein Knoten für tot erklärt wird. Ist dieser Wert zu klein gewählt, kann es zu unerwünschten Ausfällen kommen, sobald einer der beiden Knoten nicht schnell genug antwortet und sich dadurch selbst für tot erklärt. Das ist vor allem dann der Fall, wenn ein Knoten

unter hoher Last arbeitet. Ein zu groß gewählter Wert, verzögert die Übernahme der Ressourcen unnötig. In Verbindung mit der Option *warntime* findet man auf der Projektseite von Linux-HA [Kle] im FAQ-Bereich weitere Informationen, um den optimalen Wert für die *deadtime* und *warntime* zu bestimmen.

warntime `Warnzeit`

Konfiguriert die Zeit, bevor eine Meldung über das Ausbleiben der Heartbeat-Pakete eines anderen Knotens in die entsprechende Logdatei geschrieben wird.

initdead `Zeit`

Vorgabewert: `120`

Einige Systeme benötigen nach dem Systemstart eine gewisse Zeit bis sich das Netzwerk stabilisiert hat sowie andere Dienste und Ressourcen gestartet wurden, damit Heartbeat korrekt arbeiten kann. Mit dieser Option kann die Verzögerung festgelegt werden, bis der Heartbeat-Dienst die Ressourcen des Knotens übernimmt. Dieser Wert muss experimentell bestimmt werden, damit ein Knoten nicht während des Systemstarts fälschlicherweise für tot erklärt wird, weil das Netzwerksubsystem noch nicht korrekt arbeitet. Ein grober Richtwert hierfür ist die doppelte *deadtime*.

bcast `NetworkInterface`

Konfiguriert die Schnittstelle für den UDP-Broadcast Datenverkehr von Heartbeat.

mcast `NetworkInterface Mcast_Group Port TTL Loop`

Nutzt Multicast für die Kommunikation im Cluster. Die Multicast-Adresse muss aus dem Bereich 224.0.0.0/4 (Class D) stammen.

ucast `NetworkInterface IP`

Die Clusterkommunikation erfolgt über Unicast-Pakete (Punkt-zu-Punkt).

auto_failback `on|off|legacy`

Vorgabewert: `legacy`

Mit dieser Option wird festgelegt, ob die Ressourcen wieder auf ihren ursprünglichen Knoten nach einem Failover zurückgeschwenkt werden (`on`), sobald dieser wieder verfügbar ist, oder ob die Ressourcen auf dem aktuellen Knoten verbleiben, bis ein manueller Eingriff durch den Administrator erfolgt (`off`). Letztes ist empfehlenswert, um die genaue Fehlerursache zu kategorisieren und ein unnötiges Verschieben von Ressourcen zu vermeiden. Die Option *auto_failback* ist bei neueren Linux-HA Konfigurationen ab Version 2 nutzlos und wurde durch die Option *default_stickiness* ersetzt.

node `Hostname`

Die Option *node* definiert die Mitglieder eines Clusters. Die Angabe mehrerer Knoten innerhalb derselben Zeile ist möglich. Die Hostnamen müssen mit der Ausgabe des Befehls

```
# uname -n
```

übereinstimmen, damit Heartbeat korrekt arbeitet. Andernfalls führt dies zu einem Fehler beim Start.

use_logd `yes|no`

Konfiguriert die Verwendung des Logdaemons, welcher in Heartbeat integriert ist. Andernfalls (`no`) werden die Meldungen direkt in die entsprechend eingestellten Dateien weitergeleitet. Der Vorteil des Logdaemons von Heartbeat besteht darin, dass im Regelfall keine Einträge verloren gehen sollten. Auf den Entwicklerseiten ist die Nutzung des internen Logdaemons empfohlen und die Optionen für diesen Dienst in der Konfigurationsdatei `ha_logd.cf` einzustellen.

Die Auswahl an Konfigurationsoptionen ist nicht vollständig. Weitere Optionen findet man auf den Projektseiten unter [Kle] sowie [Linb].

A.5 Konfigurationsoptionen - `ldirectord.cf`

autoreload=`yes|no`

Vorgabewert: `no`

Weist `ldirectord` an, die Konfigurationsdatei kontinuierlich auf Änderungen zu überprüfen. Ist diese Option auf `yes` gesetzt, startet sich der Prozess bei Änderung der Konfigurationsdatei automatisch neu.

checkinterval=`Zeit`

Vorgabewert: `10`

Konfiguriert den Zeitintervall zwischen den Serverüberprüfungen.

checktimeout=`Zeit`

Vorgabewert: `5`

Die Option *checktimeout* gibt die Zeit (in Sekunden) an, nachdem ein Real Server für tot erklärt wird.

fallback=`IP:Port [gate|masq|ipip]`

Gibt einen belieben Server (meist localhost) an, falls alle Real Server ausgefallen sind. Oft wird damit eine Notfallseite auf dem lokalen Webserver des Load Balancers ausgeben.

logfile=`Datei|Syslog-Einrichtung`

Vorgabewert: `/var/log/ldirectord.log`

Datei, in die alle Meldungen von `ldirectord` geschrieben werden. Alternativ dazu kann der `syslogd` benutzt werden.

quiescent=`yes|no`

Vorgabewert: `yes`

Diese Option ermöglicht das Entfernen von Real Servern aus der LVS Weiterleitungstabelle, sofern diese nicht mehr erreichbar sind. Andernfalls (`no`) wird die Gewichtung (Weight) in der Weiterleitungstabelle auf 0 gesetzt. Dies führt jedoch zu einem unerwünschten Seiteneffekt bei der Nutzung von persistenten Verbindungen, in der die diese immer zum gleichen Real Server innerhalb einer vorgegeben Zeit weitergeleitet werden, solange sich dieser noch in der Weiterleitungstabelle befindet. Um diesen Verhalten zu vermeiden, muss der Befehl

```
# echo 1 > /proc/sys/net/ipv4/vs/expire_quiescent_template
```

ausgeführt werden.

Die nachfolgenden Optionen dürfen nur innerhalb des Virtual-Abschnittes verwendet werden.

checktype=`connect|external|negotiate|off|on|ping|checktimeout`
: Vorgabewert: `negotiate`
Die Option *checktype* bestimmt die Art zur Überprüfung der Server.

real=`IP:Port [gate|masq|ipip]`
: Mithilfe dieser Option werden die Real Server konfiguriert.

- `gate`: Direct Routing
- `masq`: NAT
- `ipip`: IP-Tunneling

persistent=`Zeit`
: Legt die Idle-Zeit für persistente Clientverbindungen fest. Dadurch kommuniziert derselbe Client mit dem gleichen Real Server, sofern eine erneute Anfrage innerhalb der konfigurierten Zeit eintrifft.

protocol=`tcp|udp|fwm`
: Mit dieser Option wird das Kommunikationsprotokoll eingestellt. Falls mit Firewall Marks gearbeitet wird, muss die Option auf `fwm` gesetzt werden.

request=`"URI"`
: Konfiguriert das abzurufende Objekt der Real Server.

receive=`"String"`
: Legt die Zeichenkette fest, die durch *request* angefordert wurde. Falls die Zeichenkette nicht übereinstimmt, wird der betreffende Real Server für tot erklärt.

scheduler=`Algorithmus`
: Vorgabewert: `wrr`
Durch diese Option wird der Scheduler-Algorithmus konfiguriert, mit dem die logische Paketweiterleitung zu den Real Servern erfolgt. Eine Liste aller verfügbaren Scheduler findet man auf der Manpage des `ipvsadm`.

service=`Dienst`
: Legt den zu überprüfenden Dienst für die Überwachung fest.
(Voraussetzung: *checktype*=`negotiate`)

Die vorgestellten Optionen erheben keinen Anspruch auf Vollständigkeit - es wurde ein Großteil der benötigten Optionen hinreichend erklärt. Für alle weiteren sei hier auf die Manpage von `ldirectord` verwiesen.

A.6 Benchmarkergebnisse MySQL-Cluster

Hardwareumgebung:

- Motherboard Supermicro H8DMT-INF+
- CPU AMD Quad-Core Opteron 2352 @ 2.1 GHz
- RAM 8 GB DDR2 PC667
- Festplatte Western Digital 2500YS RE 250 GB
- NIC nVidia MCP55V Pro

A.6.1 Ethernet

Config / Queries	1 mysqld
read	3.032.652
write	1.083.090
other	433.236
total	4.548.978
read per second	6.065
write per second	2.166

Tabelle A.1: Benchmark: MySQL 5.0 - MyISAM (lokal)

Config / Queries	1 mysqld, 4 ndbd
read	1.502.368
write	536.560
other	214.624
total	2.253.552
read per second	3.004
write per second	1.073

Tabelle A.2: Benchmark: MySQL-Cluster 5.0 - In-Memory Table

Die extrem schlechte Performance im Ethernet-Bereich resultiert aus den Treiberproblemen mit `forcedeth` unter Linux.

A.6.2 SCI

Queries \ Config	1 mysqld, 4 ndbd
read	4.783.142
write	1.708.265
other	683.306
total	7.174.713
read per second	9.566
write per second	3.416

Tabelle A.3: Benchmark: MySQL-Cluster 5.0 - In-Memory Table

Queries \ Config	1 mysqld, 4 ndbd
read	5.507.096
write	1.966.820
other	786.728
total	826.0644
read per second	11.014
write per second	3.933

Tabelle A.4: Benchmark: MySQL-Cluster 5.1 - In-Memory Table

Queries \ Config	1 mysqld, 4 ndbd	2 mysqld, 4 ndbd
read	5.767.048	6.869.268
write	2.059.660	2.453.310
other	823.864	981.324
total	8.650.572	10.303.902
read per second	11.534	13.738
write per second	4.119	4.906

Tabelle A.5: Benchmark: MySQL-Cluster 5.1 - Disk Data Table

Queries \ Config	2 mysqld, 4 ndbd	4 mysqld, 4 ndbd	4 mysqld, 8 ndbd
read	8.408.932	11.461.870	15.575.490
write	3.003.190	4.093.525	5.562.675
other	1.201.276	1.637.410	2.225.070
total	12.613.398	17.192.805	23.363.235
read per second	16.817	22.923	31.150
write per second	6.006	8.187	11.125

Tabelle A.6: Benchmark: MySQL-Cluster 5.1 - Disk Data Table und externes Storage

Literaturverzeichnis

[Ahn07] AHNERT, Sven: *Virtuelle Maschinen mit VMware und Microsoft. Für Entwicklung, Schulung, Test und Produktion*. Addison-Wesley, München, 2007. – ISBN 9783827323743

[AMD] AMD: *AMD Virtualization*. http://www.amd.com/us/products/technologies/virtualization, Abruf: 23. Dezember 2009. Internet

[Ana] ANANDTECH, IT @.: *Hardware Virtualization: the Nuts and Bolts*. http://www.anandtech.com/it/showdoc.aspx?i=3263, Abruf: 13. September 2009. Internet

[Aps] APSIS: *Pound: Reverse-Proxy and Load-Balancer*. http://www.apsis.ch/pound, Abruf: 28. Dezember 2009. Internet

[Bec05] BECKER, Hubert: *Einflussgrößen auf die Zuverlässigkeit*. Version: 2005. http://www.hubertbecker-online.de/log3_1_2.htm, Abruf: 21. Dezember 2009. Internet

[Bre06] BRENDEL, Jens-Christoph: *Linux-Magazin Technical Review 01: Virtualisierung*. 1. Linux New Media AG, 2006. – ISBN 9783939551034

[Bre07] BRENDEL, Jens-Christoph: *Linux-Magazin Technical Review 04: High Availability*. 4. Linux New Media AG, 2007. – ISBN 9783939551065

[cac] *Cacti: The Complete RRDTool-based Graphing Solution*. http://www.cacti.net, Abruf: 12. Januar 2010. Internet

[CC04] CLUSTER COMPUTING, IEEE Task F.: *High Availability (HA)*. Version: 2004. http://www.ieeetfcc.org/high-availability.html, Abruf: 12. September 2009. Internet

[Cis] CISCO: *Understanding Simple Network Management Protocol (SNMP) Traps*. http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a0080094aa5.shtml, Abruf: 23. Januar 2010. Internet

[cor] *Corosync*. http://www.corosync.org, Abruf: 10. Januar 2010. Internet

[Dik] DIKE, Jeff: *User-mode Linux*. http://user-mode-linux.sourceforge.net, Abruf: 24. Dezember 2009. Internet

[Fas] FASHEH, Mark: *OCFS2: The Oracle Clustered File System, Version 2*. http://oss.oracle.com/projects/ocfs2/dist/documentation/fasheh.pdf, Abruf: 04. Januar 2010. Internet

[Gru09] GRUNWALD, Lukas: Undercover - Sicherheitslücken in virtuellen Umgebungen. In: *heise iX* (2009), November, Nr. 11/2009, S. 131–133

[Hal08] HALUSCHAK, Bernhard: *Fehlertoleranter Speicher schützt vor Systemausfällen und Datenverlust*. Version: April 2008. http://www.tecchannel.de/server/hardware/402181/fehlertoleranter_speicher_schuetzt_vor_systemausfaellen_und_datenverlust, Abruf: 09. Januar 2010. Internet

[Han08] HANTELMANN, Fred: Kostenfall - Mit Virtualisierung RZ-Kosten halbieren. In: *heise iX* (2008), Dezember, Nr. 12/2008, S. 88–91

[HAP] *HAProxy: The Reliable, High Performance TCP/HTTP Load Balancer*. http://haproxy.1wt.eu, Abruf: 28. Dezember 2009. Internet

[Hel04] HELD, Andrea: *Oracle 10g Hochverfügbarkeit. Die ausfallsichere Datenbank mit RAC, Data Guard und Flashback (Edition Oracle)*. Addison-Wesley, München, 2004. – ISBN 9783827321633

[Hel09] HELSLEY, Matt: *LXC: Linux container tools*. Version: 2009. https://www.ibm.com/developerworks/linux/library/l-lxc-containers, Abruf: 24. Dezember 2009. Internet

[Hoc08] HOCHSTÄTTER, Christoph H.: *Virtualisierung mit Server-CPUs: Leistungsbremse inklusive*. Version: 12 2008. http://zdnet.de/39199764, Abruf: 13. Oktober 2009. Internet

[HRG01] HARVARD RESEARCH GROUP, Inc.: *HA Forecast*. Version: 2001. http://www.hrgresearch.com/pdf/HAS%20Forecast%20rpt%20082301%20p.pdf, Abruf: 12. September 2009. Internet

[Int06] INTEL: Intel Virtualization Technology. In: *Intel Technology Journal* 10 (2006). http://www.intel.com/technology/itj/2006/v10i3/3-xen/4-extending-with-intel-vt.htm

[Int08] INTEL: *Virtualization Technology for Directed I/O Architecture Specification*. Version: September 2008. http://download.intel.com/technology/computing/vptech/Intel%28r%29_VT_for_Direct_IO.pdf, Abruf: 23. Dezember 2009. Internet

[ITW] ITWISSEN: *Paravirtualisierung*. http://www.itwissen.info/definition/lexikon/Paravirtualisierung-para-virtualization.html, Abruf: 23. Dezember 2009. Internet

[kee] *Keepalived for Linux - Linux High Availability*. http://www.keepalived.org/, Abruf: 04. Januar 2010. Internet

[Kle] KLEINER, Dmytri: *Linux-HA*. http://www.linux-ha.org/HomePage, Abruf: 12. Oktober 2009. Internet

[Kop] KOPYTOV, Alexey: *SysBench: a system performance benchmark*. http://sysbench.sourceforge.net, Abruf: 06. Januar 2010. Internet

[KVM] *Kernel-Based Virtual Machine*. http://www.linux-kvm.org, Abruf: 06. Januar 2010. Internet

[Len09] LENZ, Ulrich: *Hochverfügbarkeit ohne Vorurteile - Zehn Mythen rund um hochverfügbare IT-Systeme*. Version: 2009. http://www.tecchannel.de/server/hardware/2020347/mythen_rund_um_hochverfuegbarkeit_ausfallsicherheit_redundanz, Abruf: 18. Dezember 2009. Internet

[LINa] *LINBIT - Your Way to High Availability*. http://www.linbit.com, Abruf: 31. Dezember 2009. Internet

[Linb] *Linux-HA Wiki*. http://www.linux-ha.org/wiki, Abruf: 28. Dezember 2009. Internet

[Linc] *Linux-Magazin: Hochverfügbarkeit*. http://www.linux-magazin.de/Themengebiete/Administration/Hochverfuegbarkeit, Abruf: 01. Januar 2010. Internet

[Lind] *Linux-Magazin: Virtualisierung*. http://www.linux-magazin.de/Themengebiete/Administration/Virtualisierung, Abruf: 01. Januar 2010. Internet

[LINe] LINBIT: *DRBD:What is DRBD*. http://www.drbd.org, Abruf: 12. November 2009. Internet

[LINf] LINBIT: *Open Source Software - csync2*. http://oss.linbit.com/csync2/, Abruf: 04. Januar 2010. Internet

[Los09] LOSCHWITZ, Martin: *Linux-Magazin: Hochverfügbarkeit mit Linux im Wandel*. Version: Dezember 2009. http://www.linux-magazin.de/Online-Artikel/Hochverfuegbarkeit-mit-Linux-im-Wandel?category=376, Abruf: 01. Januar 2010. Internet

[LVH09] LUTHER, Jörg ; VILSBECK, Christian ; HALUSCHAK, Bernhard: *RAID im Überblick*. Version: August 2009. http://www.tecchannel.de/storage/extra/401665/raid_sicherheit_level_server_storage_performance_festplatten_controller, Abruf: 09. Januar 2010. Internet

[Mac09] MACK, Joseph: *LVS: Fwmarks (firewall marks)*. Version: 2009. http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.fwmark.html, Abruf: 04. Januar 2010. Internet

[McN09] MCNIE, Nigel: *A Five Minute Guide to Linux Containers for Debian*. Version: 2009. http://nigel.mcnie.name/blog/a-five-minute-guide-to-linux-containers-for-debian, Abruf: 24. Dezember 2009. Internet

[Mica] MICROSOFT: *Grundlegendes zu Quorumkonfigurationen in einem Failovercluster*. http://technet.microsoft.com/de-de/library/cc731739.aspx, Abruf: 27. Dezember 2009. Internet

[Micb] MICROSOFT: *TechNet - Informationen zu Ausfallzeiten*. http://technet.microsoft.com/de-de/library/aa998704(EXCHG.65).aspx, Abruf: 06. November 2009. Internet

[mrt] *MRTG - The Multi Router Traffic Grapher*. http://oss.oetiker.ch/mrtg, Abruf: 12. Januar 2010. Internet

[Nag] *Nagios - The Industry Standard in IT Infrastructure Monitoring*. http://www.nagios.org, Abruf: 06. Januar 2010. Internet

[NI09] NATIONAL-INSTRUMENTS: *Wie funktioniert Virtualisierung?* Version: September 2009. http://zone.ni.com/devzone/cda/tut/p/id/10318, Abruf: 23. Dezember 2009. Internet

[opea] *OpenAIS*. http://www.openais.org, Abruf: 10. Januar 2010. Internet

[Opeb] *OpenNMS - Enterprise-grade Open-source Network Management*. http://www.opennms.org, Abruf: 06. Januar 2010. Internet

[opec] *OpenVZ*. http://www.openvz.org, Abruf: 01. Dezember 2009. Internet

[Ora] ORACLE: *Project: OCFS2*. http://oss.oracle.com/projects/ocfs2, Abruf: 04. Januar 2010. Internet

[pac] *Pacemaker - A scalable High-Availability cluster resource manager*. http://www.clusterlabs.org, Abruf: 10. Januar 2010. Internet

[Par] PARALLELS: *Betriebssystem-Virtualisierung*. http://www.parallels.com/de/products/virtuozzo/os, Abruf: 1. Dezember 2009. Internet

[PG74] POPEK, Gerald J. ; GOLDBERG, Robert P.: Formal requirements for virtualizable third generation architectures. In: *Commun. ACM* 17 (1974), Nr. 7, S. 412–421. http://dx.doi.org/10.1145/361011.361073. – DOI 10.1145/361011.361073. – ISSN 0001–0782

[Pic09] PICHT, Hans-Joachim: *XEN Kochbuch. Intelligente Virtualisierungslösungen mit XEN 3*. 1. O'Reilly, 2009. – ISBN 9783897217294

[QEM] *QEMU - open source processor emulator*. http://www.qemu.org, Abruf: 06. Januar 2010. Internet

[Reda] REDHAT: *Cluster Logical Volume Manager*. http://sources.redhat.com/cluster/clvm, Abruf: 09. Januar 2010. Internet

[Redb] REDHAT: *The Piranha Solution*. http://www.redhat.com/software/rha/cluster/piranha, Abruf: 04. Januar 2010. Internet

[RHE] *Red Hat Enterprise Virtualization*. http://www.redhat.com/virtualization/rhev, Abruf: 06. Januar 2010. Internet

[SBZD07] SPRANG, Henning ; BENK, Timo ; ZDRZALEK, Jaroslaw ; DEHNER, Ralph: *Xen: Virtualisierung unter Linux*. 1. Open Source Press, 2007. – ISBN 9783937514291

[Sch08] SCHWARTZKOPFF, Michael: *Clusterbau mit Linux-HA Version 2*. 1. O'Reilly, 2008. – ISBN 9783897217799

[Sch09] SCHWARTZKOPFF, Michael: *Clusterbau: Hochverfügbarkeit mit pacemaker, OpenAIS, heartbeat und LVS*. 2. Auflage. O'Reilly, 2009. – ISBN 9783897219199

[SM] SUN MICROSYSTEMS, Inc.: *MySQL Cluster*. http://www.mysql.de/products/database/cluster, Abruf: 12. Januar 2010. Internet

[SM09a] SUN MICROSYSTEMS, Inc.: *Bekannte Beschränkungen von MySQL Cluster*. Version: Dezember 2009. http://dev.mysql.com/doc/refman/5.1/de/mysql-cluster-limitations.html, Abruf: 07. Januar 2010. Internet

[SM09b] SUN MICROSYSTEMS, Inc.: *MySQL 5.1 Referenzhandbuch*. Version: Dezember 2009. http://dev.mysql.com/doc/refman/5.1/de, Abruf: 24. Dezember 2009. Internet

[SM09c] SUN MICROSYSTEMS, Inc.: *MySQL Cluster: Knoten, Knotengruppen, Repliken und Partitionen*. Version: Dezember 2009. http://dev.mysql.com/doc/refman/5.1/de/mysql-cluster-nodes-groups.html, Abruf: 06. Januar 2010. Internet

[SM09d] SUN MICROSYSTEMS, Inc.: *MySQL Cluster Referenzhandbuch*. Version: Dezember 2009. http://dev.mysql.com/doc/refman/5.1/de/ndbcluster.html, Abruf: 06. Januar 2010. Internet

[SM09e] SUN MICROSYSTEMS, Inc.: *Replikation bei MySQL*. Version: Dezember 2009. http://dev.mysql.com/doc/refman/5.1/de/replication.html, Abruf: 11. Januar 2010. Internet

[Som] SOMMERGUT, Wolfgang: *Virtualisierung: Anforderungen an x86-Hardware*. http://www.tecchannel.de/server/virtualisierung/2019534/virtualisierung_anforderungen_x86_hardware_prozessor_speicher_io, Abruf: 28. Oktober 2009. Internet

[Spe] SPEAKMAN, Kasey: *Open Cluster Framework*. http://www.opencf.org, Abruf: 27. Dezember 2009. Internet

[top] *TOP500 Supercomputing Sites*. http://www.top500.org, Abruf: 21. Dezember 2009. Internet

[VGa] VMWARE GLOBAL, Inc.: *A Comparison of Software and Hardware Techniques for x86 Virtualization*. http://www.vmware.com/pdf/asplos235_adams.pdf, Abruf: 13. Oktober 2009. Internet

[VGb] VMWARE GLOBAL, Inc.: *Software and Hardware Techniques for x86 Virtualization*. http://www.vmware.com/files/pdf/software_hardware_tech_x86_virt.pdf, Abruf: 13. Oktober 2009. Internet

[VGc] VMWARE GLOBAL, Inc.: *Transparente Paravirtualisierung*. http://www.vmware.com/de/interfaces/paravirtualization.html, Abruf: 23. Dezember 2009. Internet

[VG05] VMWARE GLOBAL, Inc.: *Virtualization: Architectural Considerations And Other Evaluation Criteria*. Version: 2005. http://www.vmware.com/pdf/virtualization_considerations.pdf, Abruf: 12. Oktober 2009. Internet

[VG06] VMWARE GLOBAL, Inc.: *Virtualization Overview*. Version: 2006. http://www.vmware.com/pdf/virtualization.pdf, Abruf: 30. September 2009. Internet

[VG07] VMWARE GLOBAL, Inc.: *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. Version: 2007. http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf, Abruf: 12. Oktober 2009. Internet

[VG09] VMWARE GLOBAL, Inc.: *Neuheiten bei VMware vSphere 4.0*. Version: 2009. http://www.vmware.com/de/support/vsphere4/doc/vsp_40_new_feat_de.html, Abruf: 23. Dezember 2009. Internet

[Xen] XEN: *Xen hypervisor, the powerful open source industry standard for virtualization*. http://www.xen.org, Abruf: 24. Dezember 2009. Internet

[XLB] *XLB HTTP Load Balancer*. http://sourceforge.net/projects/xlb, Abruf: 03. Januar 2010. Internet

[Zhaa] ZHANG, Wensong: *Job Scheduling Algorithms in Linux Virtual Server*. http://www.linuxvirtualserver.org/docs/scheduling.html, Abruf: 03. Januar 2010. Internet

[Zhab] ZHANG, Wensong: *KTCPVS Software - Application-Level Load Balancing*. http://www.linuxvirtualserver.org/software/ktcpvs/ktcpvs.html, Abruf: 3. Januar 2010. Internet

[Zhac] ZHANG, Wensong: *The Linux Virtual Server Project - Linux Server Cluster for Load Balancing*. http://www.linuxvirtualserver.org, Abruf: 24. Dezember 2009. Internet

[Zhad] ZHANG, Wensong: *Ultra Monkey: Load Balancing and High Availability Solution*. http://www.ultramonkey.org, Abruf: 28. Dezember 2009. Internet

[Zim06] ZIMMER, Dennis: *VMware und Microsoft Virtual Server - VMware GSX, VMware ESX und Microsoft Virtual Server im professionellen Einsatz, mit CD*. 1. Galileo Press, 2006. – ISBN 9783898427012

Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Mittweida, den 29. Januar 2010

_____

Peter Großöhme